

Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.16 Reference Manual

Brian M. Adams, William J. Bohnhoff, Robert A. Canfield, Wesley P. Coomber, Keith R. Dalbey, Mohamed S. Ebeida, John P. Eddy, Michael S. Eldred, Gianluca Geraci, Russell W. Hooper, Patricia D. Hough, Kenneth T. Hu, John D. Jakeman, Carson Kent, Mohammad Khalil, Kathryn A. Maupin, Jason A. Monschke, Teresa Portone, Elliott M. Ridgway, Ahmad A. Rushdi, D. Thomas Seidl, J. Adam Stephens, Laura P. Swiler, Anh Tran, Dena M. Vigil, Timothy M. Wildey, and Justin G. Winokur; with Friedrich Menhorn (Technical University of Munich) and Xiaoshu Zeng (University of Southern California)

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

May 12, 2022

Abstract

The Dakota toolkit provides a flexible and extensible interface between simulation codes and iterative analysis methods. Dakota contains algorithms for optimization with gradient and nongradient-based methods; uncertainty quantification with sampling, reliability, and stochastic expansion methods; parameter estimation with nonlinear least squares methods; and sensitivity/variance analysis with design of experiments and parameter study methods. These capabilities may be used on their own or as components within advanced strategies such as surrogate-based optimization, mixed integer nonlinear programming, or optimization under uncertainty. By employing object-oriented design to implement abstractions of the key components required for iterative systems analyses, the Dakota toolkit provides a flexible and extensible problem-solving environment for design and performance analysis of computational models on high performance computers.

This report serves as a reference manual for the commands specification for the Dakota software, providing input overviews, option descriptions, and example specifications.

Contents

1	Main Page	1
1.1	How to Use this Manual	1
2	Running Dakota	3
2.1	Usage	3
2.2	Examples	4
2.3	Execution Phases	5
2.4	Restarting Dakota Studies	5
2.4.1	Writing Restart Files	5
2.4.2	Using Restart Files	6
2.4.3	Appending to a Restart File	6
2.4.4	Working with multiple Restart Files	6
2.4.5	How it Works	6
2.5	The Dakota Restart Utility	6
2.5.1	Print Command	7
2.5.2	Neutral File Format	8
2.5.3	Tabular Format	8
2.5.4	Concatenation of Multiple Restart Files	9
2.5.5	Removal of Corrupted Data	10
3	Dakota HDF5 Output	11
3.1	HDF5 Concepts	11
3.2	Accessing Results	11
3.3	Organization of Results	12
3.3.1	Study Metadata	12
3.3.2	A Note about Variables Storage	13
3.3.3	Sampling Moments	13
3.3.4	Correlations	13
3.3.5	Probability Density	13
3.3.6	Level Mappings	13
3.3.7	Variance-Based Decomposition (Sobol' Indices)	14
3.3.8	Integration and Expansion Moments	14

3.3.9	Extreme Responses	14
3.3.10	Parameter Sets	14
3.3.11	Variable Slices	14
3.3.12	Best Parameters	14
3.3.13	Best Objective Functions	14
3.3.14	Best Nonlinear Constraints	14
3.3.15	Calibration	15
3.3.16	Parameter Confidence Intervals	15
3.3.17	Best Model Responses (without configuration variables)	15
3.3.18	Best Model Responses (with configuration variables)	15
3.3.19	Multistart and Pareto Set	15
3.4	Organization of Evaluations	15
3.4.1	Sources of Evaluation Data	15
3.4.2	High-level Organization of Evaluation Data	17
3.4.2.1	Interfaces	17
3.4.2.2	Models	18
3.4.3	Low-Level Organization of Evaluation Data	18
3.4.3.1	Variables	18
3.4.3.2	Responses	19
3.4.3.3	Properties	19
3.4.3.4	Metadata	19
3.4.4	Selecting Models and Interfaces to Store	20
3.5	Distribution Parameters	20
3.5.1	Naming Conventions and Layout	21
3.5.2	h5py Examples	22
3.5.3	Known Limitations	22
3.5.4	Metadata	23
3.5.5	Parameter Listing for All Types	23
4	Test Problems	25
4.1	Textbook	25
4.2	Rosenbrock	27
5	Dakota Input Specification	29
5.1	Dakota Keywords	29
5.2	Input Spec Overview	29
5.2.1	Common Specification Mistakes	29
5.2.2	Specifying Arguments	30
5.3	Sample Input Files	31
5.4	Input Spec Summary	34

6	Topics Area	117
6.1	admin	117
6.1.1	empty	117
6.1.2	problem	118
6.1.3	not_yet_reviewed	118
6.2	dakota_IO	118
6.2.1	dakota_inputs	119
6.2.1.1	block	119
6.2.1.2	block_identifier	119
6.2.1.3	block_pointer	120
6.2.1.4	data_import_capabilities	123
6.2.2	dakota_output	123
6.2.3	file_formats	124
6.3	dakota_concepts	132
6.3.1	method_independent_controls	133
6.3.2	block	139
6.3.2.1	block_identifier	139
6.3.2.2	block_pointer	139
6.3.3	strategies	143
6.3.3.1	advanced_strategies	143
6.3.4	command_line_options	143
6.3.5	restarts	143
6.3.6	pointers	144
6.3.6.1	block_pointer	144
6.3.6.2	objective_function_pointer	147
6.4	models	147
6.4.1	surrogate_models	148
6.4.2	surrogate_based_optimization_methods	151
6.4.3	recast_models	152
6.4.4	multifidelity_models	152
6.4.5	reduced_order_models	152
6.4.6	nested_models	152
6.4.7	advanced_model_recursion	152
6.4.8	hybrid_and_recursions_logic	153
6.5	variables	153
6.5.1	variable_domain	153
6.5.2	continuous_variables	153
6.5.3	discrete_variables	154
6.5.4	linear_constraints	156
6.5.5	variable_type	157

6.5.6	design_variables	157
6.5.7	aleatory_uncertain_variables	157
6.5.8	epistemic_uncertain_variables	158
6.5.9	state_variables	159
6.6	responses	159
6.6.1	response_types	160
6.6.2	nonlinear_constraints	160
6.7	interface	160
6.7.1	simulation_file_management	161
6.7.2	workflow_management	161
6.7.3	advanced_simulation_interfaces	161
6.7.4	simulation_failure	161
6.7.5	concurrency_and_parallelism	161
6.8	methods	163
6.8.1	parameter_studies	163
6.8.2	sensitivity_analysis_and_design_of_experiments	164
6.8.3	design_and_analysis_of_computer_experiments	164
6.8.4	sampling	165
6.8.5	uncertainty_quantification	165
6.8.6	aleatory_uncertainty_quantification_methods	167
6.8.7	sampling	167
6.8.8	reliability_methods	168
6.8.9	stochastic_expansion_methods	170
6.8.10	epistemic_uncertainty_quantification_methods	170
6.8.11	interval_estimation	170
6.8.12	evidence_theory	172
6.8.13	variable_support	172
6.8.14	optimization_and_calibration	174
6.8.14.1	Optimization Formulations	174
6.8.14.2	Optimizing with Dakota: Choosing a Method	177
6.8.15	local_optimization_methods	178
6.8.15.1	unconstrained	178
6.8.15.2	constrained	179
6.8.15.3	sequential_quadratic_programming	179
6.8.16	global_optimization_methods	179
6.8.17	bayesian_calibration	180
6.8.18	nonlinear_least_squares	181
6.8.19	advanced_optimization	181
6.8.19.1	scaling	182
6.8.19.2	multiobjective_methods	182

6.8.19.3	surrogate_based_optimization_methods	182
6.9	advanced_topics	182
6.9.1	advanced_strategies	183
6.9.2	advanced_model_recursion	183
6.9.3	hybrid_and_recursions_logic	183
6.9.4	advanced_simulation_interfaces	183
6.9.5	simulation_failure	183
6.9.6	concurrency_and_parallelism	184
6.9.7	advanced_optimization	185
6.9.7.1	scaling	186
6.9.7.2	multiobjective_methods	186
6.9.7.3	surrogate_based_optimization_methods	186
6.10	packages	186
6.10.1	package_coliny	187
6.10.2	package_conmin	188
6.10.3	package_ddace	188
6.10.4	package_dot	189
6.10.5	package_fsudace	189
6.10.6	package_hopspack	190
6.10.7	package_jega	190
6.10.8	package_nlpql	191
6.10.9	package_npsol	191
6.10.10	package_optpp	191
6.10.11	package_psuade	192
6.10.12	package_queso	192
6.10.13	package_scolib	193
7	Keywords Area	195
7.1	environment	196
7.1.1	tabular_data	197
7.1.1.1	tabular_data_file	198
7.1.1.2	custom_annotated	199
7.1.1.3	annotated	202
7.1.1.4	freeform	203
7.1.2	output_file	204
7.1.3	error_file	205
7.1.4	read_restart	205
7.1.4.1	stop_restart	207
7.1.5	write_restart	207
7.1.6	output_precision	208

7.1.7	results_output	209
7.1.7.1	results_output_file	209
7.1.7.2	text	210
7.1.7.3	hdf5	211
7.1.8	graphics	222
7.1.9	check	223
7.1.10	pre_run	223
7.1.10.1	input	224
7.1.10.2	output	225
7.1.11	run	231
7.1.11.1	input	232
7.1.11.2	output	232
7.1.12	post_run	233
7.1.12.1	input	233
7.1.12.2	output	239
7.1.13	top_method_pointer	240
7.2	method	241
7.2.1	id_method	250
7.2.2	output	251
7.2.2.1	debug	252
7.2.2.2	verbose	253
7.2.2.3	normal	253
7.2.2.4	quiet	253
7.2.2.5	silent	254
7.2.3	final_solutions	254
7.2.4	hybrid	255
7.2.4.1	sequential	256
7.2.4.2	embedded	263
7.2.4.3	collaborative	271
7.2.5	multi_start	277
7.2.5.1	method_name	278
7.2.5.2	method_pointer	280
7.2.5.3	random_starts	280
7.2.5.4	starting_points	282
7.2.5.5	iterator_servers	282
7.2.5.6	iterator_scheduling	283
7.2.5.7	processors_per_iterator	285
7.2.6	pareto_set	285
7.2.6.1	method_name	286
7.2.6.2	method_pointer	288

7.2.6.3	random_weight_sets	289
7.2.6.4	weight_sets	290
7.2.6.5	iterator_servers	291
7.2.6.6	iterator_scheduling	291
7.2.6.7	processors_per_iterator	293
7.2.7	branch_and_bound	294
7.2.7.1	method_pointer	296
7.2.7.2	method_name	296
7.2.7.3	scaling	298
7.2.8	surrogate_based_local	300
7.2.8.1	method_pointer	303
7.2.8.2	method_name	303
7.2.8.3	model_pointer	304
7.2.8.4	soft_convergence_limit	305
7.2.8.5	truth_surrogate_bypass	306
7.2.8.6	approx_subproblem	306
7.2.8.7	merit_function	311
7.2.8.8	acceptance_logic	314
7.2.8.9	constraint_relax	316
7.2.8.10	trust_region	317
7.2.8.11	max_iterations	321
7.2.8.12	convergence_tolerance	322
7.2.8.13	constraint_tolerance	323
7.2.9	surrogate_based_global	324
7.2.9.1	method_pointer	326
7.2.9.2	method_name	327
7.2.9.3	model_pointer	327
7.2.9.4	replace_points	329
7.2.9.5	max_iterations	329
7.2.10	dot_frog	330
7.2.10.1	max_iterations	332
7.2.10.2	convergence_tolerance	332
7.2.10.3	constraint_tolerance	333
7.2.10.4	speculative	334
7.2.10.5	max_function_evaluations	335
7.2.10.6	scaling	336
7.2.10.7	model_pointer	337
7.2.11	dot_mmfd	339
7.2.11.1	max_iterations	340
7.2.11.2	convergence_tolerance	341

7.2.11.3	constraint_tolerance	342
7.2.11.4	speculative	343
7.2.11.5	max_function_evaluations	344
7.2.11.6	scaling	344
7.2.11.7	model_pointer	346
7.2.12	dot_bfgs	348
7.2.12.1	max_iterations	349
7.2.12.2	convergence_tolerance	350
7.2.12.3	constraint_tolerance	351
7.2.12.4	speculative	352
7.2.12.5	max_function_evaluations	353
7.2.12.6	scaling	353
7.2.12.7	model_pointer	355
7.2.13	dot_slp	357
7.2.13.1	max_iterations	358
7.2.13.2	convergence_tolerance	359
7.2.13.3	constraint_tolerance	359
7.2.13.4	speculative	360
7.2.13.5	max_function_evaluations	361
7.2.13.6	scaling	362
7.2.13.7	model_pointer	364
7.2.14	dot_sqp	365
7.2.14.1	max_iterations	367
7.2.14.2	convergence_tolerance	367
7.2.14.3	constraint_tolerance	368
7.2.14.4	speculative	369
7.2.14.5	max_function_evaluations	370
7.2.14.6	scaling	371
7.2.14.7	model_pointer	372
7.2.15	conmin_frcg	374
7.2.15.1	max_iterations	375
7.2.15.2	convergence_tolerance	376
7.2.15.3	constraint_tolerance	377
7.2.15.4	speculative	378
7.2.15.5	max_function_evaluations	379
7.2.15.6	scaling	379
7.2.15.7	model_pointer	381
7.2.16	conmin_mfd	383
7.2.16.1	max_iterations	384
7.2.16.2	convergence_tolerance	385

7.2.16.3	constraint_tolerance	386
7.2.16.4	speculative	387
7.2.16.5	max_function_evaluations	388
7.2.16.6	scaling	388
7.2.16.7	model_pointer	390
7.2.17	dl_solver	392
7.2.17.1	max_function_evaluations	393
7.2.17.2	scaling	393
7.2.17.3	model_pointer	395
7.2.18	npsol_sqp	397
7.2.18.1	verify_level	399
7.2.18.2	function_precision	399
7.2.18.3	linesearch_tolerance	400
7.2.18.4	convergence_tolerance	400
7.2.18.5	max_iterations	401
7.2.18.6	constraint_tolerance	402
7.2.18.7	speculative	402
7.2.18.8	max_function_evaluations	403
7.2.18.9	scaling	404
7.2.18.10	model_pointer	406
7.2.19	nlssol_sqp	407
7.2.19.1	verify_level	409
7.2.19.2	function_precision	410
7.2.19.3	linesearch_tolerance	410
7.2.19.4	convergence_tolerance	411
7.2.19.5	max_iterations	411
7.2.19.6	constraint_tolerance	412
7.2.19.7	speculative	413
7.2.19.8	max_function_evaluations	414
7.2.19.9	scaling	414
7.2.19.10	model_pointer	416
7.2.20	nlpq1_sqp	418
7.2.20.1	max_iterations	419
7.2.20.2	convergence_tolerance	420
7.2.20.3	max_function_evaluations	421
7.2.20.4	scaling	422
7.2.20.5	model_pointer	424
7.2.21	optpp_cg	425
7.2.21.1	max_step	427
7.2.21.2	gradient_tolerance	427

7.2.21.3	max_iterations	428
7.2.21.4	convergence_tolerance	428
7.2.21.5	speculative	429
7.2.21.6	max_function_evaluations	430
7.2.21.7	scaling	431
7.2.21.8	model_pointer	433
7.2.22	optpp_q_newton	434
7.2.22.1	search_method	436
7.2.22.2	merit_function	439
7.2.22.3	steplength_to_boundary	441
7.2.22.4	centering_parameter	442
7.2.22.5	max_step	442
7.2.22.6	gradient_tolerance	443
7.2.22.7	max_iterations	443
7.2.22.8	convergence_tolerance	444
7.2.22.9	speculative	445
7.2.22.10	max_function_evaluations	446
7.2.22.11	scaling	447
7.2.22.12	model_pointer	448
7.2.23	optpp_fd_newton	450
7.2.23.1	search_method	452
7.2.23.2	merit_function	455
7.2.23.3	steplength_to_boundary	457
7.2.23.4	centering_parameter	458
7.2.23.5	max_step	458
7.2.23.6	gradient_tolerance	459
7.2.23.7	max_iterations	459
7.2.23.8	convergence_tolerance	460
7.2.23.9	speculative	461
7.2.23.10	max_function_evaluations	462
7.2.23.11	scaling	463
7.2.23.12	model_pointer	464
7.2.24	optpp_g_newton	466
7.2.24.1	search_method	468
7.2.24.2	merit_function	471
7.2.24.3	steplength_to_boundary	473
7.2.24.4	centering_parameter	474
7.2.24.5	max_step	474
7.2.24.6	gradient_tolerance	475
7.2.24.7	max_iterations	475

7.2.24.8	convergence_tolerance	476
7.2.24.9	speculative	477
7.2.24.10	max_function_evaluations	478
7.2.24.11	scaling	479
7.2.24.12	model_pointer	480
7.2.25	optpp_newton	482
7.2.25.1	search_method	484
7.2.25.2	merit_function	487
7.2.25.3	steplength_to_boundary	489
7.2.25.4	centering_parameter	490
7.2.25.5	max_step	490
7.2.25.6	gradient_tolerance	491
7.2.25.7	max_iterations	491
7.2.25.8	convergence_tolerance	492
7.2.25.9	speculative	493
7.2.25.10	max_function_evaluations	494
7.2.25.11	scaling	495
7.2.25.12	model_pointer	496
7.2.26	optpp_pds	498
7.2.26.1	search_scheme_size	499
7.2.26.2	max_iterations	500
7.2.26.3	convergence_tolerance	501
7.2.26.4	max_function_evaluations	502
7.2.26.5	scaling	502
7.2.26.6	model_pointer	504
7.2.27	demo_tpl	506
7.2.27.1	max_function_evaluations	507
7.2.27.2	max_iterations	507
7.2.27.3	convergence_tolerance	508
7.2.27.4	variable_tolerance	509
7.2.27.5	solution_target	510
7.2.27.6	options_file	511
7.2.28	rol	511
7.2.28.1	max_iterations	513
7.2.28.2	variable_tolerance	514
7.2.28.3	gradient_tolerance	515
7.2.28.4	constraint_tolerance	516
7.2.28.5	options_file	516
7.2.28.6	scaling	517
7.2.28.7	model_pointer	519

7.2.29	asynch_pattern_search	520
7.2.29.1	initial_delta	523
7.2.29.2	contraction_factor	524
7.2.29.3	variable_tolerance	525
7.2.29.4	solution_target	526
7.2.29.5	synchronization	526
7.2.29.6	merit_function	528
7.2.29.7	constraint_penalty	532
7.2.29.8	smoothing_factor	533
7.2.29.9	constraint_tolerance	533
7.2.29.10	max_function_evaluations	534
7.2.29.11	scaling	535
7.2.29.12	model_pointer	536
7.2.30	mesh_adaptive_search	538
7.2.30.1	initial_delta	541
7.2.30.2	variable_tolerance	542
7.2.30.3	function_precision	544
7.2.30.4	seed	544
7.2.30.5	history_file	545
7.2.30.6	display_format	546
7.2.30.7	variable_neighborhood_search	548
7.2.30.8	neighbor_order	549
7.2.30.9	display_all_evaluations	550
7.2.30.10	use_surrogate	552
7.2.30.11	max_iterations	557
7.2.30.12	max_function_evaluations	557
7.2.30.13	scaling	558
7.2.30.14	model_pointer	560
7.2.31	nowpac	561
7.2.31.1	trust_region	562
7.2.31.2	max_iterations	567
7.2.31.3	max_function_evaluations	568
7.2.31.4	scaling	568
7.2.31.5	model_pointer	570
7.2.32	snowpac	572
7.2.32.1	seed	573
7.2.32.2	trust_region	574
7.2.32.3	max_iterations	578
7.2.32.4	max_function_evaluations	579
7.2.32.5	scaling	580

7.2.32.6	model_pointer	581
7.2.33	moga	583
7.2.33.1	fitness_type	586
7.2.33.2	replacement_type	588
7.2.33.3	niching_type	592
7.2.33.4	convergence_type	596
7.2.33.5	postprocessor_type	599
7.2.33.6	max_iterations	600
7.2.33.7	max_function_evaluations	601
7.2.33.8	scaling	602
7.2.33.9	population_size	603
7.2.33.10	log_file	604
7.2.33.11	print_each_pop	604
7.2.33.12	initialization_type	605
7.2.33.13	crossover_type	607
7.2.33.14	mutation_type	612
7.2.33.15	seed	619
7.2.33.16	convergence_tolerance	620
7.2.33.17	model_pointer	620
7.2.34	soga	622
7.2.34.1	fitness_type	625
7.2.34.2	replacement_type	626
7.2.34.3	convergence_type	629
7.2.34.4	max_iterations	634
7.2.34.5	max_function_evaluations	634
7.2.34.6	scaling	635
7.2.34.7	population_size	637
7.2.34.8	log_file	637
7.2.34.9	print_each_pop	638
7.2.34.10	initialization_type	638
7.2.34.11	crossover_type	641
7.2.34.12	mutation_type	646
7.2.34.13	seed	651
7.2.34.14	convergence_tolerance	652
7.2.34.15	model_pointer	653
7.2.35	coliny_pattern_search	654
7.2.35.1	constant_penalty	658
7.2.35.2	no_expansion	659
7.2.35.3	expand_after_success	659
7.2.35.4	pattern_basis	660

7.2.35.5	stochastic	661
7.2.35.6	total_pattern_size	662
7.2.35.7	exploratory_moves	662
7.2.35.8	synchronization	664
7.2.35.9	contraction_factor	666
7.2.35.10	constraint_penalty	667
7.2.35.11	initial_delta	667
7.2.35.12	variable_tolerance	668
7.2.35.13	solution_target	669
7.2.35.14	seed	670
7.2.35.15	show_misc_options	671
7.2.35.16	misc_options	671
7.2.35.17	max_iterations	672
7.2.35.18	convergence_tolerance	672
7.2.35.19	max_function_evaluations	673
7.2.35.20	scaling	674
7.2.35.21	model_pointer	676
7.2.36	coliny_solis_wets	677
7.2.36.1	contract_after_failure	680
7.2.36.2	no_expansion	680
7.2.36.3	expand_after_success	681
7.2.36.4	constant_penalty	681
7.2.36.5	contraction_factor	682
7.2.36.6	constraint_penalty	682
7.2.36.7	initial_delta	683
7.2.36.8	variable_tolerance	684
7.2.36.9	solution_target	685
7.2.36.10	seed	685
7.2.36.11	show_misc_options	686
7.2.36.12	misc_options	687
7.2.36.13	max_iterations	687
7.2.36.14	convergence_tolerance	688
7.2.36.15	max_function_evaluations	689
7.2.36.16	scaling	690
7.2.36.17	model_pointer	691
7.2.37	coliny_cobyla	693
7.2.37.1	initial_delta	696
7.2.37.2	variable_tolerance	696
7.2.37.3	solution_target	697
7.2.37.4	seed	697

7.2.37.5	show_misc_options	698
7.2.37.6	misc_options	699
7.2.37.7	max_iterations	699
7.2.37.8	convergence_tolerance	700
7.2.37.9	max_function_evaluations	701
7.2.37.10	scaling	702
7.2.37.11	model_pointer	703
7.2.38	coliny_direct	705
7.2.38.1	division	708
7.2.38.2	global_balance_parameter	709
7.2.38.3	local_balance_parameter	710
7.2.38.4	max_boxsize_limit	710
7.2.38.5	min_boxsize_limit	711
7.2.38.6	constraint_penalty	711
7.2.38.7	solution_target	712
7.2.38.8	seed	712
7.2.38.9	show_misc_options	713
7.2.38.10	misc_options	713
7.2.38.11	max_iterations	714
7.2.38.12	convergence_tolerance	715
7.2.38.13	max_function_evaluations	716
7.2.38.14	scaling	716
7.2.38.15	model_pointer	718
7.2.39	coliny_ea	720
7.2.39.1	population_size	723
7.2.39.2	initialization_type	724
7.2.39.3	fitness_type	726
7.2.39.4	replacement_type	728
7.2.39.5	crossover_rate	731
7.2.39.6	crossover_type	731
7.2.39.7	mutation_rate	734
7.2.39.8	mutation_type	735
7.2.39.9	constraint_penalty	741
7.2.39.10	solution_target	742
7.2.39.11	seed	742
7.2.39.12	show_misc_options	743
7.2.39.13	misc_options	744
7.2.39.14	max_iterations	744
7.2.39.15	convergence_tolerance	745
7.2.39.16	max_function_evaluations	746

7.2.39.17	scaling	746
7.2.39.18	model_pointer	748
7.2.40	coliny_beta	750
7.2.40.1	beta_solver_name	751
7.2.40.2	solution_target	752
7.2.40.3	seed	752
7.2.40.4	show_misc_options	753
7.2.40.5	misc_options	754
7.2.40.6	max_iterations	754
7.2.40.7	convergence_tolerance	755
7.2.40.8	max_function_evaluations	756
7.2.40.9	scaling	756
7.2.40.10	model_pointer	758
7.2.41	nl2sol	760
7.2.41.1	function_precision	763
7.2.41.2	absolute_conv_tol	763
7.2.41.3	x_conv_tol	764
7.2.41.4	singular_conv_tol	764
7.2.41.5	singular_radius	765
7.2.41.6	false_conv_tol	765
7.2.41.7	initial_trust_radius	766
7.2.41.8	covariance	766
7.2.41.9	regression_diagnostics	767
7.2.41.10	convergence_tolerance	767
7.2.41.11	max_iterations	768
7.2.41.12	speculative	769
7.2.41.13	max_function_evaluations	770
7.2.41.14	scaling	771
7.2.41.15	model_pointer	772
7.2.42	nonlinear_cg	774
7.2.42.1	misc_options	775
7.2.42.2	convergence_tolerance	776
7.2.42.3	max_iterations	776
7.2.42.4	scaling	777
7.2.42.5	model_pointer	779
7.2.43	ncsu_direct	781
7.2.43.1	solution_target	783
7.2.43.2	min_boxsize_limit	783
7.2.43.3	volume_boxsize_limit	784
7.2.43.4	convergence_tolerance	784

7.2.43.5	max_iterations	785
7.2.43.6	max_function_evaluations	786
7.2.43.7	scaling	787
7.2.43.8	model_pointer	788
7.2.44	genie_opt_darts	790
7.2.44.1	seed	791
7.2.44.2	max_function_evaluations	792
7.2.44.3	scaling	792
7.2.44.4	model_pointer	794
7.2.45	genie_direct	796
7.2.45.1	seed	797
7.2.45.2	max_function_evaluations	797
7.2.45.3	scaling	798
7.2.45.4	model_pointer	800
7.2.46	efficient_global	801
7.2.46.1	initial_samples	804
7.2.46.2	seed	804
7.2.46.3	batch_size	805
7.2.46.4	max_iterations	809
7.2.46.5	convergence_tolerance	809
7.2.46.6	x_conv_tol	810
7.2.46.7	gaussian_process	810
7.2.46.8	use_derivatives	825
7.2.46.9	import_build_points_file	825
7.2.46.10	export_approx_points_file	832
7.2.46.11	model_pointer	838
7.2.47	surrogate_based_uq	840
7.2.47.1	samples_on_emulator	842
7.2.47.2	sample_type	842
7.2.47.3	rng	845
7.2.47.4	probability_refinement	847
7.2.47.5	final_moments	850
7.2.47.6	response_levels	853
7.2.47.7	probability_levels	862
7.2.47.8	reliability_levels	865
7.2.47.9	gen_reliability_levels	866
7.2.47.10	distribution	867
7.2.47.11	variance_based_decomp	870
7.2.47.12	diagonal_covariance	873
7.2.47.13	full_covariance	873

7.2.47.14	import_approx_points_file	874
7.2.47.15	export_approx_points_file	880
7.2.47.16	seed	886
7.2.47.17	fixed_seed	887
7.2.47.18	model_pointer	888
7.2.48	function_train	889
7.2.48.1	p_refinement	895
7.2.48.2	max_refinement_iterations	899
7.2.48.3	convergence_tolerance	900
7.2.48.4	metric_scale	901
7.2.48.5	regression_type	903
7.2.48.6	max_solver_iterations	905
7.2.48.7	max_cross_iterations	906
7.2.48.8	solver_tolerance	906
7.2.48.9	response_scaling	907
7.2.48.10	tensor_grid	907
7.2.48.11	collocation_points	908
7.2.48.12	collocation_ratio	909
7.2.48.13	rounding_tolerance	909
7.2.48.14	arithmetic_tolerance	910
7.2.48.15	start_order	910
7.2.48.16	adapt_order	912
7.2.48.17	kick_order	913
7.2.48.18	max_order	913
7.2.48.19	max_cv_order_candidates	914
7.2.48.20	start_rank	915
7.2.48.21	adapt_rank	915
7.2.48.22	kick_rank	916
7.2.48.23	max_rank	917
7.2.48.24	max_cv_rank_candidates	918
7.2.48.25	samples_on_emulator	918
7.2.48.26	sample_type	919
7.2.48.27	rng	921
7.2.48.28	probability_refinement	924
7.2.48.29	final_moments	927
7.2.48.30	response_levels	930
7.2.48.31	probability_levels	939
7.2.48.32	reliability_levels	941
7.2.48.33	gen_reliability_levels	943
7.2.48.34	distribution	944

7.2.48.35	variance_based_decomp	946
7.2.48.36	diagonal_covariance	949
7.2.48.37	full_covariance	950
7.2.48.38	import_approx_points_file	950
7.2.48.39	export_approx_points_file	957
7.2.48.40	seed	963
7.2.48.41	fixed_seed	964
7.2.48.42	model_pointer	964
7.2.49	multifidelity_function_train	966
7.2.49.1	p_refinement	971
7.2.49.2	max_refinement_iterations	976
7.2.49.3	convergence_tolerance	976
7.2.49.4	metric_scale	977
7.2.49.5	statistics_mode	980
7.2.49.6	allocation_control	982
7.2.49.7	discrepancy_emulation	984
7.2.49.8	rounding_tolerance	986
7.2.49.9	arithmetic_tolerance	987
7.2.49.10	regression_type	987
7.2.49.11	max_solver_iterations	990
7.2.49.12	max_cross_iterations	990
7.2.49.13	solver_tolerance	991
7.2.49.14	response_scaling	991
7.2.49.15	tensor_grid	992
7.2.49.16	collocation_points_sequence	992
7.2.49.17	collocation_ratio	993
7.2.49.18	start_order_sequence	994
7.2.49.19	adapt_order	995
7.2.49.20	kick_order	996
7.2.49.21	max_order	997
7.2.49.22	max_cv_order_candidates	997
7.2.49.23	start_rank_sequence	998
7.2.49.24	adapt_rank	999
7.2.49.25	kick_rank	1000
7.2.49.26	max_rank	1000
7.2.49.27	max_cv_rank_candidates	1001
7.2.49.28	samples_on_emulator	1002
7.2.49.29	sample_type	1002
7.2.49.30	rng	1005
7.2.49.31	probability_refinement	1007

7.2.49.32	final_moments	1010
7.2.49.33	response_levels	1014
7.2.49.34	probability_levels	1022
7.2.49.35	reliability_levels	1025
7.2.49.36	gen_reliability_levels	1026
7.2.49.37	distribution	1028
7.2.49.38	variance_based_decomp	1030
7.2.49.39	diagonal_covariance	1033
7.2.49.40	full_covariance	1033
7.2.49.41	import_approx_points_file	1034
7.2.49.42	export_approx_points_file	1040
7.2.49.43	seed_sequence	1046
7.2.49.44	fixed_seed	1047
7.2.49.45	model_pointer	1048
7.2.50	multilevel_function_train	1050
7.2.50.1	max_iterations	1055
7.2.50.2	allocation_control	1056
7.2.50.3	convergence_tolerance	1060
7.2.50.4	metric_scale	1061
7.2.50.5	discrepancy_emulation	1063
7.2.50.6	rounding_tolerance	1065
7.2.50.7	arithmetic_tolerance	1066
7.2.50.8	regression_type	1066
7.2.50.9	max_solver_iterations	1069
7.2.50.10	max_cross_iterations	1069
7.2.50.11	solver_tolerance	1070
7.2.50.12	response_scaling	1070
7.2.50.13	tensor_grid	1071
7.2.50.14	collocation_points_sequence	1071
7.2.50.15	collocation_ratio	1072
7.2.50.16	start_order_sequence	1073
7.2.50.17	adapt_order	1074
7.2.50.18	kick_order	1075
7.2.50.19	max_order	1076
7.2.50.20	max_cv_order_candidates	1076
7.2.50.21	start_rank_sequence	1077
7.2.50.22	adapt_rank	1078
7.2.50.23	kick_rank	1079
7.2.50.24	max_rank	1079
7.2.50.25	max_cv_rank_candidates	1080

7.2.50.26	<code>samples_on_emulator</code>	1081
7.2.50.27	<code>sample_type</code>	1081
7.2.50.28	<code>rng</code>	1084
7.2.50.29	<code>probability_refinement</code>	1086
7.2.50.30	<code>final_moments</code>	1089
7.2.50.31	<code>response_levels</code>	1092
7.2.50.32	<code>probability_levels</code>	1101
7.2.50.33	<code>reliability_levels</code>	1104
7.2.50.34	<code>gen_reliability_levels</code>	1105
7.2.50.35	<code>distribution</code>	1106
7.2.50.36	<code>variance_based_decomp</code>	1109
7.2.50.37	<code>diagonal_covariance</code>	1112
7.2.50.38	<code>full_covariance</code>	1112
7.2.50.39	<code>import_approx_points_file</code>	1113
7.2.50.40	<code>export_approx_points_file</code>	1119
7.2.50.41	<code>seed_sequence</code>	1125
7.2.50.42	<code>fixed_seed</code>	1126
7.2.50.43	<code>model_pointer</code>	1127
7.2.51	<code>polynomial_chaos</code>	1129
7.2.51.1	<code>p_refinement</code>	1135
7.2.51.2	<code>max_refinement_iterations</code>	1140
7.2.51.3	<code>convergence_tolerance</code>	1141
7.2.51.4	<code>metric_scale</code>	1142
7.2.51.5	<code>quadrature_order</code>	1144
7.2.51.6	<code>sparse_grid_level</code>	1146
7.2.51.7	<code>cubature_integrand</code>	1150
7.2.51.8	<code>expansion_order</code>	1151
7.2.51.9	<code>orthogonal_least_interpolation</code>	1196
7.2.51.10	<code>import_expansion_file</code>	1206
7.2.51.11	<code>askey</code>	1206
7.2.51.12	<code>wiener</code>	1207
7.2.51.13	<code>normalized</code>	1208
7.2.51.14	<code>export_expansion_file</code>	1208
7.2.51.15	<code>samples_on_emulator</code>	1209
7.2.51.16	<code>sample_type</code>	1209
7.2.51.17	<code>rng</code>	1212
7.2.51.18	<code>probability_refinement</code>	1214
7.2.51.19	<code>final_moments</code>	1217
7.2.51.20	<code>response_levels</code>	1220
7.2.51.21	<code>probability_levels</code>	1229

7.2.51.22	reliability_levels	1232
7.2.51.23	gen_reliability_levels	1233
7.2.51.24	distribution	1234
7.2.51.25	variance_based_decomp	1237
7.2.51.26	diagonal_covariance	1240
7.2.51.27	full_covariance	1240
7.2.51.28	import_approx_points_file	1241
7.2.51.29	export_approx_points_file	1247
7.2.51.30	seed	1253
7.2.51.31	fixed_seed	1254
7.2.51.32	model_pointer	1255
7.2.52	multifidelity_polynomial_chaos	1256
7.2.52.1	p_refinement	1262
7.2.52.2	max_refinement_iterations	1267
7.2.52.3	convergence_tolerance	1268
7.2.52.4	metric_scale	1268
7.2.52.5	statistics_mode	1270
7.2.52.6	allocation_control	1272
7.2.52.7	discrepancy_emulation	1274
7.2.52.8	quadrature_order_sequence	1276
7.2.52.9	sparse_grid_level_sequence	1279
7.2.52.10	expansion_order_sequence	1282
7.2.52.11	orthogonal_least_interpolation	1314
7.2.52.12	askey	1324
7.2.52.13	wiener	1325
7.2.52.14	normalized	1325
7.2.52.15	export_expansion_file	1326
7.2.52.16	samples_on_emulator	1326
7.2.52.17	sample_type	1327
7.2.52.18	rng	1329
7.2.52.19	probability_refinement	1332
7.2.52.20	final_moments	1335
7.2.52.21	response_levels	1338
7.2.52.22	probability_levels	1347
7.2.52.23	reliability_levels	1349
7.2.52.24	gen_reliability_levels	1351
7.2.52.25	distribution	1352
7.2.52.26	variance_based_decomp	1354
7.2.52.27	diagonal_covariance	1357
7.2.52.28	full_covariance	1358

7.2.52.29	import_approx_points_file	1358
7.2.52.30	export_approx_points_file	1365
7.2.52.31	seed_sequence	1371
7.2.52.32	fixed_seed	1372
7.2.52.33	model_pointer	1373
7.2.53	multilevel_polynomial_chaos	1374
7.2.53.1	max_iterations	1379
7.2.53.2	allocation_control	1380
7.2.53.3	convergence_tolerance	1383
7.2.53.4	metric_scale	1384
7.2.53.5	discrepancy_emulation	1386
7.2.53.6	expansion_order_sequence	1388
7.2.53.7	orthogonal_least_interpolation	1420
7.2.53.8	askey	1430
7.2.53.9	wiener	1430
7.2.53.10	normalized	1431
7.2.53.11	export_expansion_file	1432
7.2.53.12	samples_on_emulator	1432
7.2.53.13	sample_type	1433
7.2.53.14	rng	1435
7.2.53.15	probability_refinement	1437
7.2.53.16	final_moments	1440
7.2.53.17	response_levels	1443
7.2.53.18	probability_levels	1452
7.2.53.19	reliability_levels	1455
7.2.53.20	gen_reliability_levels	1456
7.2.53.21	distribution	1457
7.2.53.22	variance_based_decomp	1460
7.2.53.23	diagonal_covariance	1463
7.2.53.24	full_covariance	1463
7.2.53.25	import_approx_points_file	1464
7.2.53.26	export_approx_points_file	1470
7.2.53.27	seed_sequence	1476
7.2.53.28	fixed_seed	1477
7.2.53.29	model_pointer	1478
7.2.54	stoch_collocation	1480
7.2.54.1	p_refinement	1486
7.2.54.2	h_refinement	1490
7.2.54.3	max_refinement_iterations	1495
7.2.54.4	convergence_tolerance	1495

7.2.54.5	metric_scale	1496
7.2.54.6	quadrature_order	1498
7.2.54.7	sparse_grid_level	1501
7.2.54.8	piecewise	1506
7.2.54.9	askey	1507
7.2.54.10	wiener	1507
7.2.54.11	use_derivatives	1508
7.2.54.12	samples_on_emulator	1508
7.2.54.13	sample_type	1509
7.2.54.14	rng	1511
7.2.54.15	probability_refinement	1514
7.2.54.16	final_moments	1517
7.2.54.17	response_levels	1520
7.2.54.18	probability_levels	1529
7.2.54.19	reliability_levels	1531
7.2.54.20	gen_reliability_levels	1533
7.2.54.21	distribution	1534
7.2.54.22	variance_based_decomp	1536
7.2.54.23	diagonal_covariance	1539
7.2.54.24	full_covariance	1540
7.2.54.25	import_approx_points_file	1540
7.2.54.26	export_approx_points_file	1547
7.2.54.27	seed	1553
7.2.54.28	fixed_seed	1554
7.2.54.29	model_pointer	1554
7.2.55	multifidelity_stoch_collocation	1556
7.2.55.1	p_refinement	1561
7.2.55.2	h_refinement	1565
7.2.55.3	max_refinement_iterations	1569
7.2.55.4	convergence_tolerance	1569
7.2.55.5	metric_scale	1570
7.2.55.6	statistics_mode	1573
7.2.55.7	allocation_control	1575
7.2.55.8	discrepancy_emulation	1577
7.2.55.9	quadrature_order_sequence	1579
7.2.55.10	sparse_grid_level_sequence	1581
7.2.55.11	piecewise	1586
7.2.55.12	askey	1587
7.2.55.13	wiener	1588
7.2.55.14	use_derivatives	1588

7.2.55.15	<code>samples_on_emulator</code>	1589
7.2.55.16	<code>sample_type</code>	1589
7.2.55.17	<code>rng</code>	1592
7.2.55.18	<code>probability_refinement</code>	1594
7.2.55.19	<code>final_moments</code>	1597
7.2.55.20	<code>response_levels</code>	1600
7.2.55.21	<code>probability_levels</code>	1609
7.2.55.22	<code>reliability_levels</code>	1612
7.2.55.23	<code>gen_reliability_levels</code>	1613
7.2.55.24	<code>distribution</code>	1614
7.2.55.25	<code>variance_based_decomp</code>	1617
7.2.55.26	<code>diagonal_covariance</code>	1620
7.2.55.27	<code>full_covariance</code>	1620
7.2.55.28	<code>import_approx_points_file</code>	1621
7.2.55.29	<code>export_approx_points_file</code>	1627
7.2.55.30	<code>seed_sequence</code>	1633
7.2.55.31	<code>fixed_seed</code>	1634
7.2.55.32	<code>model_pointer</code>	1635
7.2.56	<code>sampling</code>	1637
7.2.56.1	<code>samples</code>	1641
7.2.56.2	<code>seed</code>	1642
7.2.56.3	<code>fixed_seed</code>	1643
7.2.56.4	<code>sample_type</code>	1643
7.2.56.5	<code>refinement_samples</code>	1647
7.2.56.6	<code>d_optimal</code>	1649
7.2.56.7	<code>variance_based_decomp</code>	1651
7.2.56.8	<code>backfill</code>	1653
7.2.56.9	<code>principal_components</code>	1654
7.2.56.10	<code>wilks</code>	1657
7.2.56.11	<code>final_moments</code>	1662
7.2.56.12	<code>response_levels</code>	1665
7.2.56.13	<code>probability_levels</code>	1674
7.2.56.14	<code>reliability_levels</code>	1677
7.2.56.15	<code>gen_reliability_levels</code>	1678
7.2.56.16	<code>distribution</code>	1679
7.2.56.17	<code>rng</code>	1682
7.2.56.18	<code>model_pointer</code>	1684
7.2.57	<code>multilevel_sampling</code>	1686
7.2.57.1	<code>seed_sequence</code>	1689
7.2.57.2	<code>fixed_seed</code>	1690

7.2.57.3	pilot_samples	1691
7.2.57.4	solution_mode	1692
7.2.57.5	sample_type	1694
7.2.57.6	export_sample_sequence	1697
7.2.57.7	allocation_target	1702
7.2.57.8	qoi_aggregation	1710
7.2.57.9	convergence_tolerance	1713
7.2.57.10	convergence_tolerance_type	1713
7.2.57.11	convergence_tolerance_target	1715
7.2.57.12	max_iterations	1716
7.2.57.13	max_function_evaluations	1717
7.2.57.14	final_statistics	1717
7.2.57.15	rng	1725
7.2.57.16	model_pointer	1727
7.2.58	multifidelity_sampling	1728
7.2.58.1	seed_sequence	1732
7.2.58.2	fixed_seed	1733
7.2.58.3	pilot_samples	1734
7.2.58.4	solution_mode	1735
7.2.58.5	numerical_solve	1737
7.2.58.6	sample_type	1741
7.2.58.7	export_sample_sequence	1743
7.2.58.8	convergence_tolerance	1750
7.2.58.9	max_iterations	1751
7.2.58.10	max_function_evaluations	1751
7.2.58.11	final_statistics	1752
7.2.58.12	rng	1760
7.2.58.13	model_pointer	1763
7.2.59	multilevel_multifidelity_sampling	1764
7.2.59.1	seed_sequence	1767
7.2.59.2	fixed_seed	1768
7.2.59.3	pilot_samples	1769
7.2.59.4	solution_mode	1769
7.2.59.5	sample_type	1772
7.2.59.6	export_sample_sequence	1774
7.2.59.7	convergence_tolerance	1781
7.2.59.8	max_iterations	1782
7.2.59.9	max_function_evaluations	1783
7.2.59.10	final_statistics	1783
7.2.59.11	rng	1791

7.2.59.12	model_pointer	1794
7.2.60	approximate_control_variate	1795
7.2.60.1	acv_independent_sampling	1799
7.2.60.2	acv_multifidelity	1800
7.2.60.3	acv_kl	1800
7.2.60.4	pilot_samples	1801
7.2.60.5	solution_mode	1801
7.2.60.6	truth_fixed_by_pilot	1804
7.2.60.7	sqp	1804
7.2.60.8	nip	1805
7.2.60.9	seed_sequence	1805
7.2.60.10	fixed_seed	1806
7.2.60.11	sample_type	1807
7.2.60.12	export_sample_sequence	1810
7.2.60.13	convergence_tolerance	1816
7.2.60.14	max_iterations	1817
7.2.60.15	max_function_evaluations	1818
7.2.60.16	final_statistics	1819
7.2.60.17	rng	1827
7.2.60.18	model_pointer	1829
7.2.61	importance_sampling	1830
7.2.61.1	samples	1833
7.2.61.2	seed	1834
7.2.61.3	import	1835
7.2.61.4	adapt_import	1835
7.2.61.5	mm_adapt_import	1836
7.2.61.6	refinement_samples	1836
7.2.61.7	max_iterations	1837
7.2.61.8	convergence_tolerance	1838
7.2.61.9	response_levels	1838
7.2.61.10	probability_levels	1847
7.2.61.11	gen_reliability_levels	1849
7.2.61.12	distribution	1851
7.2.61.13	rng	1853
7.2.61.14	model_pointer	1855
7.2.62	gpais	1857
7.2.62.1	build_samples	1859
7.2.62.2	seed	1860
7.2.62.3	samples_on_emulator	1860
7.2.62.4	import_build_points_file	1861

7.2.62.5	export_approx_points_file	1868
7.2.62.6	max_iterations	1874
7.2.62.7	response_levels	1875
7.2.62.8	probability_levels	1883
7.2.62.9	gen_reliability_levels	1886
7.2.62.10	distribution	1887
7.2.62.11	rng	1889
7.2.62.12	model_pointer	1892
7.2.63	adaptive_sampling	1893
7.2.63.1	initial_samples	1896
7.2.63.2	seed	1897
7.2.63.3	samples_on_emulator	1898
7.2.63.4	fitness_metric	1898
7.2.63.5	batch_selection	1900
7.2.63.6	refinement_samples	1903
7.2.63.7	import_build_points_file	1904
7.2.63.8	export_approx_points_file	1911
7.2.63.9	misc_options	1916
7.2.63.10	max_iterations	1917
7.2.63.11	response_levels	1918
7.2.63.12	probability_levels	1926
7.2.63.13	gen_reliability_levels	1928
7.2.63.14	distribution	1930
7.2.63.15	rng	1932
7.2.63.16	model_pointer	1934
7.2.64	pof_darts	1936
7.2.64.1	build_samples	1937
7.2.64.2	seed	1938
7.2.64.3	lipschitz	1939
7.2.64.4	samples_on_emulator	1940
7.2.64.5	response_levels	1941
7.2.64.6	probability_levels	1949
7.2.64.7	gen_reliability_levels	1952
7.2.64.8	distribution	1953
7.2.64.9	rng	1956
7.2.64.10	model_pointer	1958
7.2.65	rkd_darts	1959
7.2.65.1	build_samples	1961
7.2.65.2	seed	1962
7.2.65.3	lipschitz	1963

7.2.65.4	samples_on_emulator	1964
7.2.65.5	response_levels	1964
7.2.65.6	probability_levels	1970
7.2.65.7	gen_reliability_levels	1973
7.2.65.8	distribution	1974
7.2.65.9	rng	1977
7.2.65.10	model_pointer	1978
7.2.66	global_evidence	1979
7.2.66.1	samples	1982
7.2.66.2	seed	1982
7.2.66.3	sbgo	1983
7.2.66.4	ego	2003
7.2.66.5	ea	2028
7.2.66.6	lhs	2029
7.2.66.7	response_levels	2029
7.2.66.8	probability_levels	2038
7.2.66.9	gen_reliability_levels	2040
7.2.66.10	distribution	2042
7.2.66.11	rng	2044
7.2.66.12	model_pointer	2046
7.2.67	global_interval_est	2048
7.2.67.1	samples	2050
7.2.67.2	seed	2051
7.2.67.3	max_iterations	2051
7.2.67.4	convergence_tolerance	2052
7.2.67.5	max_function_evaluations	2053
7.2.67.6	sbgo	2054
7.2.67.7	ego	2073
7.2.67.8	ea	2103
7.2.67.9	lhs	2103
7.2.67.10	rng	2104
7.2.67.11	model_pointer	2106
7.2.68	bayes_calibration	2108
7.2.68.1	queso	2111
7.2.68.2	gpmsa	2421
7.2.68.3	wasabi	2458
7.2.68.4	dream	2746
7.2.68.5	muq	3037
7.2.68.6	experimental_design	3056
7.2.68.7	calibrate_error_multipliers	3067

7.2.68.8	burn_in_samples	3072
7.2.68.9	posterior_stats	3072
7.2.68.10	chain_diagnostics	3078
7.2.68.11	model_evidence	3080
7.2.68.12	model_discrepancy	3083
7.2.68.13	sub_sampling_period	3114
7.2.68.14	probability_levels	3115
7.2.68.15	convergence_tolerance	3118
7.2.68.16	max_iterations	3118
7.2.68.17	model_pointer	3119
7.2.68.18	scaling	3121
7.2.69	dace	3122
7.2.69.1	grid	3125
7.2.69.2	random	3125
7.2.69.3	oas	3126
7.2.69.4	lhs	3126
7.2.69.5	oa_lhs	3127
7.2.69.6	box_behnken	3128
7.2.69.7	central_composite	3128
7.2.69.8	samples	3129
7.2.69.9	seed	3129
7.2.69.10	fixed_seed	3130
7.2.69.11	main_effects	3131
7.2.69.12	quality_metrics	3131
7.2.69.13	variance_based_decomp	3133
7.2.69.14	symbols	3135
7.2.69.15	model_pointer	3135
7.2.70	fsu_cvt	3137
7.2.70.1	samples	3139
7.2.70.2	seed	3140
7.2.70.3	fixed_seed	3140
7.2.70.4	latinize	3141
7.2.70.5	quality_metrics	3142
7.2.70.6	variance_based_decomp	3143
7.2.70.7	trial_type	3145
7.2.70.8	num_trials	3147
7.2.70.9	max_iterations	3148
7.2.70.10	model_pointer	3149
7.2.71	psuade_moat	3150
7.2.71.1	partitions	3152

7.2.71.2	samples	3152
7.2.71.3	seed	3153
7.2.71.4	model_pointer	3154
7.2.72	local_evidence	3155
7.2.72.1	sqp	3157
7.2.72.2	nip	3157
7.2.72.3	response_levels	3158
7.2.72.4	probability_levels	3166
7.2.72.5	gen_reliability_levels	3168
7.2.72.6	distribution	3170
7.2.72.7	model_pointer	3172
7.2.73	local_interval_est	3174
7.2.73.1	sqp	3175
7.2.73.2	nip	3176
7.2.73.3	convergence_tolerance	3176
7.2.73.4	model_pointer	3177
7.2.74	local_reliability	3178
7.2.74.1	mpp_search	3181
7.2.74.2	response_levels	3195
7.2.74.3	probability_levels	3204
7.2.74.4	reliability_levels	3206
7.2.74.5	gen_reliability_levels	3208
7.2.74.6	distribution	3209
7.2.74.7	max_iterations	3211
7.2.74.8	convergence_tolerance	3212
7.2.74.9	final_moments	3213
7.2.74.10	model_pointer	3216
7.2.75	global_reliability	3218
7.2.75.1	initial_samples	3220
7.2.75.2	x_gaussian_process	3221
7.2.75.3	u_gaussian_process	3221
7.2.75.4	surfpack	3222
7.2.75.5	dakota	3228
7.2.75.6	experimental	3229
7.2.75.7	import_build_points_file	3235
7.2.75.8	export_approx_points_file	3242
7.2.75.9	use_derivatives	3248
7.2.75.10	seed	3248
7.2.75.11	rng	3249
7.2.75.12	response_levels	3252

7.2.75.13	probability_levels	3260
7.2.75.14	gen_reliability_levels	3262
7.2.75.15	distribution	3264
7.2.75.16	max_iterations	3266
7.2.75.17	convergence_tolerance	3267
7.2.75.18	model_pointer	3268
7.2.76	fsu_quasi_mc	3269
7.2.76.1	halton	3272
7.2.76.2	hammersley	3273
7.2.76.3	latinize	3273
7.2.76.4	quality_metrics	3274
7.2.76.5	variance_based_decomp	3275
7.2.76.6	samples	3277
7.2.76.7	fixed_sequence	3277
7.2.76.8	sequence_start	3278
7.2.76.9	sequence_leap	3278
7.2.76.10	prime_base	3279
7.2.76.11	max_iterations	3280
7.2.76.12	model_pointer	3281
7.2.77	vector_parameter_study	3282
7.2.77.1	final_point	3284
7.2.77.2	step_vector	3285
7.2.77.3	num_steps	3286
7.2.77.4	model_pointer	3286
7.2.78	list_parameter_study	3288
7.2.78.1	list_of_points	3291
7.2.78.2	import_points_file	3291
7.2.78.3	model_pointer	3298
7.2.79	centered_parameter_study	3299
7.2.79.1	step_vector	3301
7.2.79.2	steps_per_variable	3302
7.2.79.3	model_pointer	3303
7.2.80	multidim_parameter_study	3304
7.2.80.1	partitions	3306
7.2.80.2	model_pointer	3307
7.2.81	richardson_extrap	3309
7.2.81.1	estimate_order	3310
7.2.81.2	converge_order	3311
7.2.81.3	converge_qoi	3311
7.2.81.4	refinement_rate	3312

7.2.81.5	convergence_tolerance	3312
7.2.81.6	max_iterations	3313
7.2.81.7	model_pointer	3314
7.3	model	3315
7.3.1	id_model	3318
7.3.2	single	3319
7.3.2.1	interface_pointer	3320
7.3.2.2	solution_level_control	3321
7.3.2.3	solution_level_cost	3322
7.3.2.4	cost_recovery_metadata	3322
7.3.3	surrogate	3323
7.3.3.1	id_surrogates	3327
7.3.3.2	global	3328
7.3.3.3	multipoint	3529
7.3.3.4	local	3532
7.3.3.5	hierarchical	3535
7.3.3.6	non_hierarchical	3543
7.3.4	nested	3545
7.3.4.1	optional_interface_pointer	3546
7.3.4.2	sub_method_pointer	3549
7.3.5	active_subspace	3643
7.3.5.1	truth_model_pointer	3645
7.3.5.2	initial_samples	3646
7.3.5.3	sample_type	3647
7.3.5.4	truncation_method	3649
7.3.5.5	dimension	3659
7.3.5.6	bootstrap_samples	3659
7.3.5.7	build_surrogate	3660
7.3.5.8	normalization	3661
7.3.6	adapted_basis	3663
7.3.6.1	truth_model_pointer	3665
7.3.6.2	truncation_tolerance	3671
7.3.7	random_field	3672
7.3.7.1	build_source	3674
7.3.7.2	expansion_form	3678
7.3.7.3	expansion_bases	3680
7.3.7.4	truncation_tolerance	3681
7.3.7.5	propagation_model_pointer	3681
7.3.8	variables_pointer	3682
7.3.9	responses_pointer	3682

7.3.10	hierarchical_tagging	3683
7.4	variables	3684
7.4.1	id_variables	3690
7.4.2	active	3691
7.4.2.1	all	3693
7.4.2.2	design	3693
7.4.2.3	uncertain	3694
7.4.2.4	aleatory	3694
7.4.2.5	epistemic	3695
7.4.2.6	state	3695
7.4.3	mixed	3695
7.4.4	relaxed	3696
7.4.5	continuous_design	3697
7.4.5.1	initial_point	3698
7.4.5.2	lower_bounds	3699
7.4.5.3	upper_bounds	3699
7.4.5.4	scale_types	3699
7.4.5.5	scales	3700
7.4.5.6	descriptors	3701
7.4.6	discrete_design_range	3702
7.4.6.1	initial_point	3702
7.4.6.2	lower_bounds	3703
7.4.6.3	upper_bounds	3704
7.4.6.4	descriptors	3704
7.4.7	discrete_design_set	3704
7.4.7.1	integer	3705
7.4.7.2	string	3712
7.4.7.3	real	3718
7.4.8	normal_uncertain	3724
7.4.8.1	means	3726
7.4.8.2	std_deviations	3726
7.4.8.3	lower_bounds	3727
7.4.8.4	upper_bounds	3727
7.4.8.5	initial_point	3727
7.4.8.6	descriptors	3728
7.4.9	lognormal_uncertain	3729
7.4.9.1	lambdas	3730
7.4.9.2	means	3732
7.4.9.3	lower_bounds	3734
7.4.9.4	upper_bounds	3734

7.4.9.5	initial_point	3735
7.4.9.6	descriptors	3735
7.4.10	uniform_uncertain	3736
7.4.10.1	lower_bounds	3737
7.4.10.2	upper_bounds	3737
7.4.10.3	initial_point	3738
7.4.10.4	descriptors	3738
7.4.11	loguniform_uncertain	3739
7.4.11.1	lower_bounds	3740
7.4.11.2	upper_bounds	3740
7.4.11.3	initial_point	3741
7.4.11.4	descriptors	3741
7.4.12	triangular_uncertain	3742
7.4.12.1	modes	3743
7.4.12.2	lower_bounds	3743
7.4.12.3	upper_bounds	3744
7.4.12.4	initial_point	3744
7.4.12.5	descriptors	3745
7.4.13	exponential_uncertain	3745
7.4.13.1	betas	3747
7.4.13.2	initial_point	3748
7.4.13.3	descriptors	3748
7.4.14	beta_uncertain	3749
7.4.14.1	alphas	3750
7.4.14.2	betas	3751
7.4.14.3	lower_bounds	3751
7.4.14.4	upper_bounds	3751
7.4.14.5	initial_point	3752
7.4.14.6	descriptors	3752
7.4.15	gamma_uncertain	3753
7.4.15.1	alphas	3754
7.4.15.2	betas	3755
7.4.15.3	initial_point	3755
7.4.15.4	descriptors	3756
7.4.16	gumbel_uncertain	3756
7.4.16.1	alphas	3758
7.4.16.2	betas	3759
7.4.16.3	initial_point	3759
7.4.16.4	descriptors	3760
7.4.17	frechet_uncertain	3760

7.4.17.1	alphas	3762
7.4.17.2	betas	3763
7.4.17.3	initial_point	3763
7.4.17.4	descriptors	3764
7.4.18	weibull_uncertain	3764
7.4.18.1	alphas	3766
7.4.18.2	betas	3766
7.4.18.3	initial_point	3767
7.4.18.4	descriptors	3768
7.4.19	histogram_bin_uncertain	3768
7.4.19.1	pairs_per_variable	3770
7.4.19.2	abscissas	3771
7.4.19.3	ordinates	3771
7.4.19.4	counts	3772
7.4.19.5	initial_point	3772
7.4.19.6	descriptors	3773
7.4.20	poisson_uncertain	3773
7.4.20.1	lambdas	3775
7.4.20.2	initial_point	3776
7.4.20.3	descriptors	3776
7.4.21	binomial_uncertain	3777
7.4.21.1	probability_per_trial	3778
7.4.21.2	num_trials	3778
7.4.21.3	initial_point	3779
7.4.21.4	descriptors	3780
7.4.22	negative_binomial_uncertain	3780
7.4.22.1	probability_per_trial	3781
7.4.22.2	num_trials	3782
7.4.22.3	initial_point	3782
7.4.22.4	descriptors	3783
7.4.23	geometric_uncertain	3784
7.4.23.1	probability_per_trial	3784
7.4.23.2	initial_point	3785
7.4.23.3	descriptors	3786
7.4.24	hypergeometric_uncertain	3786
7.4.24.1	total_population	3788
7.4.24.2	selected_population	3788
7.4.24.3	num_drawn	3789
7.4.24.4	initial_point	3790
7.4.24.5	descriptors	3791

7.4.25	histogram_point_uncertain	3791
7.4.25.1	integer	3793
7.4.25.2	string	3796
7.4.25.3	real	3800
7.4.26	uncertain_correlation_matrix	3803
7.4.27	continuous_interval_uncertain	3805
7.4.27.1	num_intervals	3807
7.4.27.2	interval_probabilities	3807
7.4.27.3	lower_bounds	3808
7.4.27.4	upper_bounds	3808
7.4.27.5	initial_point	3809
7.4.27.6	descriptors	3809
7.4.28	discrete_interval_uncertain	3810
7.4.28.1	num_intervals	3811
7.4.28.2	interval_probabilities	3812
7.4.28.3	lower_bounds	3812
7.4.28.4	upper_bounds	3813
7.4.28.5	initial_point	3813
7.4.28.6	descriptors	3814
7.4.29	discrete_uncertain_set	3814
7.4.29.1	integer	3815
7.4.29.2	string	3820
7.4.29.3	real	3824
7.4.30	continuous_state	3829
7.4.30.1	initial_state	3830
7.4.30.2	lower_bounds	3831
7.4.30.3	upper_bounds	3831
7.4.30.4	descriptors	3832
7.4.31	discrete_state_range	3832
7.4.31.1	initial_state	3833
7.4.31.2	lower_bounds	3834
7.4.31.3	upper_bounds	3834
7.4.31.4	descriptors	3835
7.4.32	discrete_state_set	3835
7.4.32.1	integer	3836
7.4.32.2	string	3840
7.4.32.3	real	3843
7.4.33	linear_inequality_constraint_matrix	3847
7.4.34	linear_inequality_lower_bounds	3849
7.4.35	linear_inequality_upper_bounds	3849

7.4.36	linear_inequality_scale_types	3850
7.4.37	linear_inequality_scales	3851
7.4.38	linear_equality_constraint_matrix	3852
7.4.39	linear_equality_targets	3853
7.4.40	linear_equality_scale_types	3854
7.4.41	linear_equality_scales	3855
7.5	interface	3856
7.5.1	id_interface	3859
7.5.2	analysis_drivers	3859
7.5.2.1	input_filter	3862
7.5.2.2	output_filter	3863
7.5.2.3	system	3864
7.5.2.4	fork	3875
7.5.2.5	direct	3886
7.5.2.6	plugin	3888
7.5.2.7	matlab	3889
7.5.2.8	python	3890
7.5.2.9	legacy_python	3891
7.5.2.10	scilab	3892
7.5.2.11	grid	3892
7.5.2.12	analysis_components	3893
7.5.3	algebraic_mappings	3893
7.5.4	failure_capture	3894
7.5.4.1	abort	3895
7.5.4.2	retry	3895
7.5.4.3	recover	3896
7.5.4.4	continuation	3896
7.5.5	deactivate	3897
7.5.5.1	active_set_vector	3898
7.5.5.2	evaluation_cache	3899
7.5.5.3	strict_cache_equality	3899
7.5.5.4	restart_file	3900
7.5.6	batch	3901
7.5.6.1	size	3902
7.5.7	asynchronous	3902
7.5.7.1	evaluation_concurrency	3903
7.5.7.2	local_evaluation_scheduling	3904
7.5.7.3	analysis_concurrency	3906
7.5.8	evaluation_servers	3907
7.5.9	evaluation_scheduling	3907

7.5.9.1	master	3908
7.5.9.2	peer	3908
7.5.10	processors_per_evaluation	3910
7.5.11	analysis_servers	3911
7.5.12	analysis_scheduling	3912
7.5.12.1	master	3912
7.5.12.2	peer	3913
7.6	responses	3913
7.6.1	id_responses	3916
7.6.2	descriptors	3917
7.6.3	objective_functions	3918
7.6.3.1	sense	3920
7.6.3.2	primary_scale_types	3921
7.6.3.3	primary_scales	3921
7.6.3.4	weights	3922
7.6.3.5	nonlinear_inequality_constraints	3923
7.6.3.6	nonlinear_equality_constraints	3926
7.6.3.7	scalar_objectives	3929
7.6.3.8	field_objectives	3930
7.6.4	calibration_terms	3933
7.6.4.1	scalar_calibration_terms	3935
7.6.4.2	field_calibration_terms	3936
7.6.4.3	primary_scales	3939
7.6.4.4	weights	3940
7.6.4.5	calibration_data	3941
7.6.4.6	calibration_data_file	3952
7.6.4.7	simulation_variance	3962
7.6.4.8	nonlinear_inequality_constraints	3963
7.6.4.9	nonlinear_equality_constraints	3966
7.6.5	response_functions	3969
7.6.5.1	scalar_responses	3970
7.6.5.2	field_responses	3971
7.6.6	no_gradients	3973
7.6.7	analytic_gradients	3974
7.6.8	mixed_gradients	3975
7.6.8.1	id_numerical_gradients	3976
7.6.8.2	id_analytic_gradients	3977
7.6.8.3	method_source	3977
7.6.8.4	dakota	3978
7.6.8.5	vendor	3981

7.6.8.6	interval_type	3981
7.6.8.7	forward	3982
7.6.8.8	central	3982
7.6.8.9	fd_step_size	3983
7.6.9	numerical_gradients	3983
7.6.9.1	method_source	3984
7.6.9.2	dakota	3985
7.6.9.3	vendor	3988
7.6.9.4	interval_type	3988
7.6.9.5	forward	3989
7.6.9.6	central	3989
7.6.9.7	fd_step_size	3990
7.6.10	no_hessians	3990
7.6.11	numerical_hessians	3991
7.6.11.1	fd_step_size	3993
7.6.11.2	relative	3993
7.6.11.3	absolute	3994
7.6.11.4	bounds	3994
7.6.11.5	forward	3995
7.6.11.6	central	3995
7.6.12	quasi_hessians	3995
7.6.12.1	bfgs	3996
7.6.12.2	sr1	3998
7.6.13	analytic_hessians	3998
7.6.14	mixed_hessians	3999
7.6.14.1	id_numerical_hessians	4000
7.6.14.2	relative	4002
7.6.14.3	absolute	4002
7.6.14.4	bounds	4003
7.6.14.5	forward	4003
7.6.14.6	central	4003
7.6.14.7	id_quasi_hessians	4004
7.6.14.8	id_analytic_hessians	4007
7.6.15	metadata	4007

Bibliographic References

4009

Chapter 1

Main Page

The Dakota software (<http://dakota.sandia.gov/>) delivers advanced parametric analysis techniques enabling quantification of margins and uncertainty, risk analysis, model calibration, and design exploration with computational models. Its methods include optimization, uncertainty quantification, parameter estimation, and sensitivity analysis, which may be used individually or as components within surrogate-based and other advanced strategies.

Author

Brian M. Adams, William J. Bohnhoff, Robert A. Canfield, Wesley P. Coomber, Keith R. Dalbey, Mohamed S. Ebeida, John P. Eddy, Michael S. Eldred, Gianluca Geraci, Russell W. Hooper, Patricia D. Hough, Kenneth T. Hu, John D. Jakeman, Carson Kent, Mohammad Khalil, Kathryn A. Maupin, Jason A. Monschke, Teresa Portone, Elliott M. Ridgway, Ahmad A. Rushdi, D. Thomas Seidl, J. Adam Stephens, Laura P. Swiler, Anh Tran, Dena M. Vigil, Timothy M. Wildey, and Justin G. Winokur; with Friedrich Menhorn and Xiaoshu Zeng

The Reference Manual documents all the input keywords that can appear in a Dakota input file to configure a Dakota study. Its organization closely mirrors the structure of `dakota.input.summary`. For more information see [Dakota Input Specification](#). For information on software structure, refer to the Developers Manual [3], and for a tour of Dakota features and capabilities, including a tutorial, refer to the User's Manual [5].

Dakota 6.14 and newer include a refreshed Examples Library, ranging from examples of Dakota input files and basic studies through complete case studies. These can be found in binary distributions in `dakota/share/dakota/examples/official` (`dakota/dakota-examples` in source distributions). Legacy examples, including many from the Dakota software manuals appear in other directories in `dakota/share/dakota/examples` (`dakota/examples` for source).

1.1 How to Use this Manual

- To learn how to run Dakota from the command line, see [Running Dakota](#)
- To learn to how to restart Dakota studies, see [Restarting Dakota Studies](#)
- To learn about the Dakota restart utility, see [The Dakota Restart Utility](#)

To find more information about a specific keyword

1. Use the search box at the top right (currently only finds keyword names)
2. Browse the Keywords tree on the left navigation pane
3. Look at the [Dakota Input Specification](#)
4. Navigate through the keyword pages, starting from the [Keywords Area](#)

To find more information about a Dakota related topic

1. Browse the Topics Area on the left navigation pane
2. Navigate through the topics pages, starting from the [Topics Area](#)

A small number of examples are included (see [Sample Input Files](#)) along with a description of the test problems (see [Test Problems](#)).

A bibliography for the Reference Manual is provided in [Bibliographic References](#)

Chapter 2

Running Dakota

The Dakota executable file is named `dakota` (`dakota.exe` on Windows) and is most commonly run from a terminal or command prompt.

2.1 Usage

If the `dakota` command is entered at the command prompt without any arguments, a usage message similar to the following appears:

```
usage: dakota [options and <args>]
  -help (Print this summary)
  -version (Print DAKOTA version number)
  -input <$val> (REQUIRED DAKOTA input file $val)
  -preproc [$val] (Pre-process input file with pyprepro or tool $val)
  -output <$val> (Redirect DAKOTA standard output to file $val)
  -error <$val> (Redirect DAKOTA standard error to file $val)
  -parser <$val> (Parsing technology: nidr[strict][:dumpfile])
  -no_input_echo (Do not echo DAKOTA input file)
  -check (Perform input checks)
  -pre_run [$val] (Perform pre-run (variables generation) phase)
  -run [$val] (Perform run (model evaluation) phase)
  -post_run [$val] (Perform post-run (final results) phase)
  -read_restart [$val] (Read an existing DAKOTA restart file $val)
  -stop_restart <$val> (Stop restart file processing at restart record $val)
  -write_restart [$val] (Write a new DAKOTA restart file $val)
```

Of these command line options, only `input` is required, and the `-input` switch can be omitted if the input file name is the final item appearing on the command line (see Examples); all other command-line inputs are optional.

- `help` prints the usage message above.
- `version` prints version information for the executable.
- `check` invokes a dry-run mode in which the input file is processed and checked for errors, but the study is not performed.
- `input` provides the name of the Dakota input file, which can optionally be pre-processed as a template using the `preproc` option.
- `output` and `error` options provide file names for redirection of the Dakota standard output (`stdout`) and standard error (`stderr`), respectively.
- The `parser` option is for debugging and will not be further described here.

- By default, Dakota will echo the input file to the output stream, but `no_input_echo` can override this behavior.
- `read_restart` and `write_restart` commands provide the names of restart databases to read from and write to, respectively.
- `stop_restart` command limits the number of restart records read from the restart database (the default is all the records) for those cases in which some evaluations were erroneous or corrupted. Restart records are recorded in order of completion. Restart management is an important technique for retaining data from expensive engineering applications.
- `-pre_run`, `-run`, and `-post_run` instruct Dakota to run one or more execution phases, excluding others. The commands must be followed by filenames as described in [Execution Phases](#).

Command line switches can be abbreviated so long as the abbreviation is unique, so the following are valid, unambiguous specifications: `-h`, `-v`, `-c`, `-i`, `-o`, `-e`, `-s`, `-w`, `-re`, `-ru`, and `-po` and can be used in place of the longer forms of the command line options.

For information on restarting Dakota, see [Restarting Dakota Studies](#) and [The Dakota Restart Utility](#).

2.2 Examples

To run Dakota with a particular input file, the following syntax can be used:

```
dakota -i dakota.in
```

or more simply

```
dakota dakota.in
```

This will echo the standard output (stdout) and standard error (stderr) messages to the terminal. To redirect stdout and stderr to separate files, the `-o` and `-e` command line options may be used:

```
dakota -i dakota.in -o dakota.out -e dakota.err
```

or

```
dakota -o dakota.out -e dakota.err dakota.in
```

Alternatively, any of a variety of Unix redirection variants can be used. Refer to [6] for more information on Unix redirection. The simplest of these redirects stdout to another file:

```
dakota dakota.in > dakota.out
```

The specified Dakota input file may instead be an dprepro/aprepro-style template file to be pre-processed prior to running Dakota. For example it might contain template expressions in curly braces:

```
# {MyLB = 2.0} {MyUB = 8.6}
variables
  uniform_uncertain 3
  upper_bounds {MyUB} {MyUB} {MyUB}
  lower_bounds {MyLB} {MyLB} {MyLB}
```

(See the Interfaces chapter in the Dakota User's Manual [?] for more information and use cases.) To pre-process the input file, specify the `preproc` flag which generates an intermediate temporary input file for use in Dakota. If Dakota's `pyprepro.py` utility is not available on the execution PATH and/or additional pre-processing options are needed, the tool location and syntax can be specified, for example:

```
# Assumes pyprepro.py is on PATH:
dakota -i dakota_rosen.tmpl -preproc

# Specify path/name of pre-processor:
dakota -i dakota_rosen.tmpl \
  -preproc "/home/user/dakota/bin/pyprepro"

# Specify Python interpreter to use, for example on Windows
dakota -i dakota_rosen.tmpl -preproc "C:/python27/python.exe \
  C:/dakota/6.10/bin/pyprepro/pyprepro.py"

# Specify additional options to pyprepro, e.g., include file:
dakota -i dakota_rosen.tmpl -preproc "pyprepro.py -I default.params"
```

2.3 Execution Phases

Dakota has three execution phases: `pre-run`, `run`, and `post-run`.

- `pre-run` can be used to generate variable sets
- `run` (core run) invokes the simulation to evaluate variables, producing responses
- `post-run` accepts variable/response sets and analyzes the results (for example, calculate correlations from a set of samples). Currently only two modes are supported and only for sampling, parameter study, and DACE methods:

(1) `pre-run` only with optional tabular output of variables:

```
dakota -i dakota.in -pre_run [::myvariables.dat]
```

(2) `post-run` only with required tabular input of variables/responses:

```
dakota -i dakota.in -post_run myvarsresponses.dat::
```

2.4 Restarting Dakota Studies

Dakota is often used to solve problems that require repeatedly running computationally expensive simulation codes. In some cases you may want to repeat an optimization study, but with a tighter final convergence tolerance. This would be costly if the entire optimization analysis had to be repeated. Interruptions imposed by computer usage policies, power outages, and system failures could also result in costly delays. However, Dakota automatically records the variable and response data from all function evaluations so that new executions of Dakota can pick up where previous executions left off. The Dakota restart file (`dakota.rst` by default) archives the tabulated interface evaluations in a binary format. The primary restart commands at the command line are `-read_restart`, `-write_restart`, and `-stop_restart`.

2.4.1 Writing Restart Files

To write a restart file using a particular name, the `-write_restart` command line input (may be abbreviated as `-w`) is used:

```
dakota -i dakota.in -write_restart my_restart_file
```

If no `-write_restart` specification is used, then Dakota will still write a restart file, but using the default name `dakota.rst` instead of a user-specified name.

To turn restart recording off, the user may use the `restart_file` keyword, in the `interface` block. This can increase execution speed and reduce disk storage requirements, but at the expense of a loss in the ability to recover and continue a run that terminates prematurely. This option is not recommended when function evaluations are costly or prone to failure. Please note that using the `deactivate restart_file` specification will result in a zero length restart file with the default name `dakota.rst`, which can overwrite an existing file.

2.4.2 Using Restart Files

To restart Dakota from a restart file, the `-read_restart` command line input (may be abbreviated as `-r`) is used:

```
dakota -i dakota.in -read_restart my_restart_file
```

If no `-read_restart` specification is used, then Dakota will not read restart information from any file (i.e., the default is no restart processing).

To read in only a portion of a restart file, the `-stop_restart` control (may be abbreviated as `-s`) is used to specify the number of entries to be read from the database. Note that this integer value corresponds to the restart record processing counter (as can be seen when using the `print` utility (see [The Dakota Restart Utility](#)) which may differ from the evaluation numbers used in the previous run if, for example, any duplicates were detected (since these duplicates are not recorded in the restart file). In the case of a `-stop_restart` specification, it is usually desirable to specify a new restart file using `-write_restart` so as to remove the records of erroneous or corrupted function evaluations. For example, to read in the first 50 evaluations from `dakota.rst`:

```
dakota -i dakota.in -r dakota.rst -s 50 -w dakota_new.rst
```

The `dakota_new.rst` file will contain the 50 processed evaluations from `dakota.rst` as well as any new evaluations. All evaluations following the 50th in `dakota.rst` have been removed from the latest restart record.

2.4.3 Appending to a Restart File

If the `-write_restart` and `-read_restart` specifications identify the same file (including the case where `-write_restart` is not specified and `-read_restart` identifies `dakota.rst`), then new evaluations will be appended to the existing restart file.

2.4.4 Working with multiple Restart Files

If the `-write_restart` and `-read_restart` specifications identify different files, then the evaluations read from the file identified by `-read_restart` are first written to the `-write_restart` file. Any new evaluations are then appended to the `-write_restart` file. In this way, restart operations can be chained together indefinitely with the assurance that all of the relevant evaluations are present in the latest restart file.

2.4.5 How it Works

Dakota's restart algorithm relies on its duplicate detection capabilities. Processing a restart file populates the list of function evaluations that have been performed. Then, when the study is restarted, it is started from the beginning (not a warm start) and many of the function evaluations requested by the iterator are intercepted by the duplicate detection code. This approach has the primary advantage of restoring the complete state of the iteration (including the ability to correctly detect subsequent duplicates) for all methods/iterators without the need for iterator-specific restart code. However, the possibility exists for numerical round-off error to cause a divergence between the evaluations performed in the previous and restarted studies. This has been rare in practice.

2.5 The Dakota Restart Utility

The Dakota restart utility program provides a variety of facilities for managing restart files from Dakota executions. The executable program name is `dakota_restart_util` and it has the following options, as shown by the usage message returned when executing the utility without any options:

Usage:

```
dakota_restart_util command <arg1> [<arg2> <arg3> ...] --options
dakota_restart_util print <restart_file>
dakota_restart_util to_neutral <restart_file> <neutral_file>
dakota_restart_util from_neutral <neutral_file> <restart_file>
dakota_restart_util to_tabular <restart_file> <text_file>
```



```

    [--custom_annotated [header] [eval_id] [interface_id]]
    [--output_precision <int>]
    dakota_restart_util remove <double> <old_restart_file> <new_restart_file>
    dakota_restart_util remove_ids <int_1> ... <int_n> <old_restart_file> <new_restart_file>
    dakota_restart_util cat <restart_file_1> ... <restart_file_n> <new_restart_file>
options:
--help                show dakota_restart_util help message
--custom_annotated arg    tabular file options: header, eval_id,
                           interface_id
--freeform            tabular file: freeform format
--output_precision arg (=10) set tabular output precision

```

Several of these functions involve format conversions. In particular, the binary format used for restart files can be converted to ASCII text and printed to the screen, converted to and from a neutral file format, or converted to a tabular format for importing into 3rd-party plotting programs. In addition, a restart file with corrupted data can be repaired by value or id, and multiple restart files can be combined to create a master database.

2.5.1 Print Command

The `print` option is useful to show contents of a restart file, since the binary format is not convenient for direct inspection. The restart data is printed in full precision, so that exact matching of points is possible for restarted runs or corrupted data removals. For example, the following command

```

dakota_restart_util print
dakota.rst

```

results in output similar to the following:

```

-----
Restart record    1  (evaluation id    1):
-----
Parameters:
                1.8000000000000000e+00 intake_dia
                1.0000000000000000e+00 flatness

Active response data:
Active set vector = { 3 3 3 3 }
                -2.4355973813420619e+00 obj_fn
                -4.7428486677140930e-01 nln_ineq_con_1
                -4.5000000000000001e-01 nln_ineq_con_2
                1.3971143170299741e-01 nln_ineq_con_3
[ -4.3644298963447897e-01  1.4999999999999999e-01 ] obj_fn gradient
[  1.3855136437818300e-01  0.0000000000000000e+00 ] nln_ineq_con_1 gradient
[  0.0000000000000000e+00  1.4999999999999999e-01 ] nln_ineq_con_2 gradient
[  0.0000000000000000e+00 -1.9485571585149869e-01 ] nln_ineq_con_3 gradient

-----
Restart record    2  (evaluation id    2):
-----
Parameters:
                2.1640000000000001e+00 intake_dia
                1.71699994018008317e+00 flatness

Active response data:
Active set vector = { 3 3 3 3 }
                -2.4869127192988878e+00 obj_fn
                6.9256958799989843e-01 nln_ineq_con_1
                -3.4245008972987528e-01 nln_ineq_con_2
                8.7142207937157910e-03 nln_ineq_con_3
[ -4.3644298963447897e-01  1.4999999999999999e-01 ] obj_fn gradient
[  2.9814239699997572e+01  0.0000000000000000e+00 ] nln_ineq_con_1 gradient
[  0.0000000000000000e+00  1.4999999999999999e-01 ] nln_ineq_con_2 gradient
[  0.0000000000000000e+00 -1.6998301774282701e-01 ] nln_ineq_con_3 gradient

...<snip>...

Restart file processing completed: 11 evaluations retrieved.

```

2.5.2 Neutral File Format

A Dakota restart file can be converted to a neutral file format using a command like the following:

```
dakota_restart_util to_neutral dakota.rst dakota.neu
```

which results in a report similar to the following:

```
Writing neutral file dakota.neu
Restart file processing completed: 11 evaluations retrieved.
```

Similarly, a neutral file can be returned to binary format using a command like the following:

```
dakota_restart_util from_neutral dakota.neu dakota.rst
```

which results in a report similar to the following:

```
Reading neutral file dakota.neu
Writing new restart file dakota.rst
Neutral file processing completed: 11 evaluations retrieved.
```

The contents of the generated neutral file are similar to the following (from the first two records for the Cylinder example in [5]).

```
6 7 2 1.8000000000000000e+00 intake_dia 1.0000000000000000e+00 flatness 0 0 0 0
NULL 4 2 1 0 3 3 3 3 1 2 obj_fn nln_ineq_con_1 nln_ineq_con_2 nln_ineq_con_3
-2.4355973813420619e+00 -4.7428486677140930e-01 -4.5000000000000001e-01
1.3971143170299741e-01 -4.3644298963447897e-01 1.4999999999999999e-01
1.3855136437818300e-01 0.0000000000000000e+00 0.0000000000000000e+00
1.4999999999999999e-01 0.0000000000000000e+00 -1.9485571585149869e-01 1
6 7 2 2.1640000000000001e+00 intake_dia 1.7169994018008317e+00 flatness 0 0 0 0
NULL 4 2 1 0 3 3 3 3 1 2 obj_fn nln_ineq_con_1 nln_ineq_con_2 nln_ineq_con_3
-2.4869127192988878e+00 6.9256958799989843e-01 -3.4245008972987528e-01
8.7142207937157910e-03 -4.3644298963447897e-01 1.4999999999999999e-01
2.9814239699997572e+01 0.0000000000000000e+00 0.0000000000000000e+00
1.4999999999999999e-01 0.0000000000000000e+00 -1.6998301774282701e-01 2
```

This format is not intended for direct viewing (`print` should be used for this purpose). Rather, the neutral file capability has been used in the past for managing portability of restart data across platforms (recent use of more portable binary formats has largely eliminated this need) or for advanced repair of restart records (in cases where the `remove` command was insufficient).

2.5.3 Tabular Format

Conversion of a binary restart file to a tabular format enables convenient import of this data into 3rd-party post-processing tools such as Matlab, TECplot, Excel, etc. This facility is nearly identical to the output activated by the `tabular_data` keyword in the Dakota input file specification, but with two important differences:

1. No function evaluations are suppressed as they are with `tabular_data` (i.e., any internal finite difference evaluations are included).
2. The conversion can be performed later, i.e., for Dakota runs executed previously.

An example command for converting a restart file to tabular format is:

```
dakota_restart_util to_tabular dakota.rst dakota.m
```

which results in a report similar to the following:

```
Writing tabular text file dakota.m
Restart file processing completed: 10 evaluations tabulated.
```

The contents of the generated tabular file are similar to the following (from the example in the Restart section of [5]). Note that while evaluations resulting from numerical derivative offsets would be reported (as described above), derivatives returned as part of the evaluations are not reported (since they do not readily fit within a compact tabular format):

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID          0.9          1.1          0.0002          0.26          0.76
2          NO_ID          0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID          0.89991      1.1 0.0002003604863 0.2598380081 0.760045
4          NO_ID          0.9          1.10011 0.0002004407265 0.259945 0.7602420121
5          NO_ID          0.9          1.09989 0.0001995607255 0.260055 0.7597580121
6          NO_ID          0.58256179 0.4772224441 0.1050555937 0.1007670171 -0.06353963386
7          NO_ID          0.5826200462 0.4772224441 0.1050386469 0.1008348962 -0.06356876195
8          NO_ID          0.5825035339 0.4772224441 0.1050725476 0.1006991449 -0.06351050577
9          NO_ID          0.58256179 0.4772701663 0.1050283245 0.100743156 -0.06349408333
10         NO_ID          0.58256179 0.4771747219 0.1050828704 0.1007908783 -0.06358517983
...
```

Controlling tabular format: The command-line options `--freeform` and `--custom_annotated` give control of headers in the resulting tabular file. Freeform will generate a tabular file with no leading row nor columns (variable and response values only). Custom annotated format accepts any or all of the options:

- `header`: include %-commented header row with labels
- `eval_id`: include leading column with evaluation ID
- `interface_id`: include leading column with interface ID

For example, to recover Dakota 6.0 tabular format, which contained a header row, leading column with evaluation ID, but no interface ID:

```
dakota_restart_util to_tabular dakota.rst dakota.m --custom_annotated header eval_id
```

Resulting in

```
%eval_id          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          0.9          1.1          0.0002          0.26          0.76
2          0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

Finally, `--output_precision integer` will generate tabular output with the specified integer digits of precision.

2.5.4 Concatenation of Multiple Restart Files

In some instances, it is useful to combine restart files into a single master function evaluation database. For example, when constructing a data fit surrogate model, data from previous studies can be pulled in and reused to create a combined data set for the surrogate fit. An example command for concatenating multiple restart files is:

```
dakota_restart_util cat dakota.rst.1 dakota.rst.2 dakota.rst.3 dakota.rst.all
```

which results in a report similar to the following:

```
Writing new restart file dakota.rst.all
dakota.rst.1 processing completed: 10 evaluations retrieved.
dakota.rst.2 processing completed: 110 evaluations retrieved.
dakota.rst.3 processing completed: 65 evaluations retrieved.
```

The `dakota.rst.all` database now contains 185 evaluations and can be read in for use in a subsequent Dakota study using the `-read_restart` option to the `dakota` executable.

2.5.5 Removal of Corrupted Data

On occasion, a simulation or computer system failure may cause a corruption of the Dakota restart file. For example, a simulation crash may result in failure of a post-processor to retrieve meaningful data. If 0's (or other erroneous data) are returned from the user's `analysis_driver`, then this bad data will get recorded in the restart file. If there is a clear demarcation of where corruption initiated (typical in a process with feedback, such as gradient-based optimization), then use of the `-stop_restart` option for the `dakota` executable can be effective in continuing the study from the point immediately prior to the introduction of bad data. If, however, there are interspersed corruptions throughout the restart database (typical in a process without feedback, such as sampling), then the `remove` and `remove_ids` options of `dakota_restart_util` can be useful.

An example of the command syntax for the `remove` option is:

```
dakota_restart_util remove 2.e-04 dakota.rst dakota.rst.repaired
```

which results in a report similar to the following:

```
Writing new restart file dakota.rst.repaired
Restart repair completed: 65 evaluations retrieved, 2 removed, 63 saved.
```

where any evaluations in `dakota.rst` having an active response function value that matches `2.e-04` within machine precision are discarded when creating `dakota.rst.repaired`.

An example of the command syntax for the `remove_ids` option is:

```
dakota_restart_util remove_ids 12 15 23 44 57 dakota.rst dakota.rst.repaired
```

which results in a report similar to the following:

```
Writing new restart file dakota.rst.repaired
Restart repair completed: 65 evaluations retrieved, 5 removed, 60 saved.
```

where evaluation ids 12, 15, 23, 44, and 57 have been discarded when creating `dakota.rst.repaired`. An important detail is that, unlike the `-stop_restart` option which operates on restart record numbers, the `remove_ids` option operates on evaluation ids. Thus, removal is not necessarily based on the order of appearance in the restart file. This distinction is important when removing restart records for a run that contained either asynchronous or duplicate evaluations, since the restart insertion order and evaluation ids may not correspond in these cases (asynchronous evaluations have ids assigned in the order of job creation but are inserted in the restart file in the order of job completion, and duplicate evaluations are not recorded which introduces offsets between evaluation id and record number). This can also be important if removing records from a concatenated restart file, since the same evaluation id could appear more than once. In this case, all evaluation records with ids matching the `remove_ids` list will be removed.

If neither of these removal options is sufficient to handle a particular restart repair need, then the fallback position is to resort to direct editing of a neutral file to perform the necessary modifications. <!-------

----->

Chapter 3

Dakota HDF5 Output

Beginning with release 6.9, Dakota gained the ability to write many method results such as the correlation matrices computed by [sampling](#) studies and the best parameters discovered by optimization methods to disk in **HDF5**. In Dakota 6.10 and above, evaluation data (variables and responses for each model or interface evaluation) may also be written. Many users may find this newly supported format more convenient than scraping or copying and pasting from Dakota's console output.

To enable HDF5 output, the [results_output](#) keyword with the [hdf5](#) option must be added to the Dakota input file. In addition, Dakota must have been built with HDF5 support. Beginning with Dakota 6.10, HDF5 is enabled in our publicly available downloads. HDF5 support is considered a somewhat experimental feature. The results of some Dakota methods are not yet written to HDF5, and in a few, limited situations, enabling HDF5 will cause Dakota to crash.

3.1 HDF5 Concepts

HDF5 is a format that is widely used in scientific software for efficiently storing and organizing data. The HDF5 standard and libraries are maintained by the [HDF Group](#).

In HDF5, data are stored in multidimensional arrays called *datasets*. Datasets are organized hierarchically in *groups*, which also can contain other groups. Datasets and groups are conceptually similar to files and directories in a filesystem. In fact, every HDF5 file contains at least one group, the root group, denoted "/", and groups and datasets are referred to using slash-delimited absolute or relative paths, which are more accurately called *link names*.

HDF5 has as one goal that data be "self-documenting" through the use of metadata. Dakota output files include two kinds of metadata.

- **Dimension Scales.** Each dimension of a dataset may have zero or more scales, which are themselves datasets. Scales are often used to provide, for example, labels analogous to column headings in a table (see the dimension scales that Dakota applies to [moments](#)) or numerical values of an independent variable (user-specified probability levels in [level mappings](#)).
- **Attributes.** key:value pairs that annotate a group or dataset. A key is always a character string, such as `dakota_version`, and (in Dakota output) the value can be a string-, integer-, or real-valued scalar. Dakota stores the number of samples that were requested in a [sampling](#) study in the attribute 'samples'.

3.2 Accessing Results

Many popular programming languages have support, either natively or from a third-party library, for reading and writing HDF5 files. The HDF Group itself supports C/C++ and Java libraries. The Dakota Project suggests the `h5py` module for Python. Examples that demonstrate using `h5py` to access and use Dakota HDF5 output may be found in the Dakota installation at `dakota/share/dakota/examples/official/hdf5`.

3.3 Organization of Results

Currently, complete or nearly complete coverage of results from sampling, optimization and calibration methods, parameter studies, and stochastic expansions exists. Coverage will continue to expand in future releases to include not only the results of all methods, but other potentially useful information such as interface evaluations and model transformations.

Methods in Dakota have a character string `Id` and are executed by Dakota one or more times. (Methods are executed more than once in studies that include a [nested model](#), for example.) The `Id` may be provided by the user in the input file using the `id_method` keyword, or it may be automatically generated by Dakota. Dakota uses the label `NO_METHOD_ID` for methods that are specified in the input file without an `id_method`, and `NOSPEC_METHOD_ID_ID_<N>` for methods that it generates for its own internal use. The `<N>` in the latter case is an incrementing integer that begins at 1.

The results for the `<N>`th execution of a method that has the label `<method Id>` are stored in the group

```
/methods/<method Id>/results/execution:<N>/
```

The `/methods` group is always present in Dakota HDF5 files, provided at least one method added results to the output. (In a future Dakota release, the top level groups `/interfaces` and `/models` will be added.) The group `execution:1` also is always present, even if there is only a single execution.

The groups and datasets for each type of result that Dakota is currently capable of storing are described in the following sections. Every dataset is documented in its own table. These tables include:

- A brief *description* of the dataset.
- The *location* of the dataset relative to `/methods/<method Id>/execution:<N>`. This path may include both literal text that is always present and replacement text. Replacement text is *<enclosed in angle brackets and italicized>*. Two examples of replacement text are *<response descriptor>* and *<variable descriptor>*, which indicate that the name of a Dakota response or variable makes up a portion of the path.
- Clarifying *notes*, where appropriate.
- The *type* (String, Integer, or Real) of the information in the dataset.
- The *shape* of the dataset; that is, the number of dimensions and the size of each dimension.
- A description of the dataset's *scales*, which includes
 - The *dimension* of the dataset that the scale belongs to.
 - The *type* (String, Integer, or Real) of the information in the scale.
 - The *label* or name of the scale.
 - The *contents* of the scale. Contents that appear in plaintext are literal and will always be present in a scale. Italicized text describes content that varies.
 - *notes* that provide further clarification about the scale.
- A description of the dataset's *attributes*, which are key:value pairs that provide helpful context for the dataset.

The **Expected Output** section of each [method](#)'s keyword documentation indicates the kinds of output, if any, that method currently can write to HDF5. These are typically in the form of bulleted lists with clarifying notes that refer back to the sections that follow.

3.3.1 Study Metadata

Several pieces of information about the Dakota study are stored as attributes of the top-level HDF5 root group (`"/`). These include:

3.3.2 A Note about Variables Storage

Variables in most Dakota output (e.g. tabular data files) and input (e.g. imported data to construct surrogates) are listed in "input spec" order. (The [variables](#) keyword section is arranged by input spec order.) In this ordering, they are sorted first by function:

1. Design
2. Aleatory
3. Epistemic
4. State

And within each of these categories, they are sorted by domain:

1. Continuous
2. Discrete integer (sets and ranges)
3. Discrete string
4. Discrete real

A shortcoming of HDF5 is that datasets are homogeneous; for example, string- and real-valued data cannot readily be stored in the same dataset. As a result, Dakota has chosen to flip "input spec" order for HDF5 and sort first by domain, then by function when storing variable information. When applicable, there may be as many as four datasets to store variable information: one to store continuous variables, another to store discrete integer variables, and so on. Within each of these, variables will be ordered by function.

3.3.3 Sampling Moments

[sampling](#) produces moments (e.g. mean, standard deviation or variance) of all responses, as well as 95% lower and upper confidence intervals for the 1st and 2nd moments. These are stored as described below. When [sampling](#) is used in incremental mode by specifying [refinement_samples](#), all results, including the `moments` group, are placed within groups named `increment:<N>`, where `<N>` indicates the increment number beginning with 1.

3.3.4 Correlations

A few different methods produce information about the correlations between pairs of variables and responses (collectively: factors). The four tables in this section describe how correlation information is stored. One important note is that HDF5 has no special, native type for symmetric matrices, and so the simple correlations and simple rank correlations are stored in dense 2D datasets.

3.3.5 Probability Density

Some aleatory UQ methods estimate the probability density of responses.

3.3.6 Level Mappings

Aleatory UQ methods can calculate level mappings (from user-specified probability, reliability, or generalized reliability to response, or vice versa).

3.3.7 Variance-Based Decomposition (Sobol' Indices)

Dakota's [sampling](#) method can produce main and total effects; stochastic expansions ([polynomial_chaos](#), [stoch_collocation](#)) additionally can produce interaction effects.

Each order (pair, 3-way, 4-way, etc) of interaction is stored in a separate dataset. The scales are unusual in that they are two-dimensional to contain the labels of the variables that participate in each interaction.

3.3.8 Integration and Expansion Moments

Stochastic expansion methods can obtain moments two ways.

3.3.9 Extreme Responses

[sampling](#) with epistemic variables produces extreme values (minimum and maximum) for each response.

3.3.10 Parameter Sets

All parameter studies ([vector_parameter_study](#), [list_parameter_study](#), [multidim_parameter_study](#), [centered_parameter_study](#)) record tables of evaluations (parameter-response pairs), similar to Dakota's tabular output file. Centered parameter studies additionally store evaluations in an order that is more natural to interpret, which is described below.

In the tabular-like listing, variables are stored according to the scheme described in a [previous section](#).

3.3.11 Variable Slices

Centered parameter studies store "slices" of the tabular data that make evaluating the effects of each variable on each response more convenient. The steps for each individual variable, including the initial or center point, and corresponding responses are stored in separate groups.

3.3.12 Best Parameters

Dakota's optimization and calibration methods report the parameters at the best point (or points, for multiple [final solutions](#)) discovered. These are stored using the scheme described in the [variables](#) section. When more than one solution is reported, the best parameters are nested in groups named `set : <N>`, where <N> is a integer numbering the set and beginning with 1.

State (and other inactive variables) are reported when using [objective functions](#) and for some [calibration](#) studies. However, when using configuration variables in a calibration, state variables are suppressed.

3.3.13 Best Objective Functions

Dakota's optimization methods report the objective functions at the best point (or points, for multiple [final solutions](#)) discovered. When more than one solution is reported, the best objective functions are nested in groups named `set : <N>`, where <N> is a integer numbering the set and beginning with 1.

3.3.14 Best Nonlinear Constraints

Dakota's optimization and calibration methods report the nonlinear constraints at the best point (or points, for multiple [final solutions](#)) discovered. When more than one solution is reported, the best constraints are nested in groups named `set : <N>`, where N is a integer numbering the set and beginning with 1.

3.3.15 Calibration

When using [calibration terms](#) with an optimization method, or when using a nonlinear least squares method such as [nl2sol](#), Dakota reports residuals and residual norms for the best point (or points, for multiple [final solutions](#)) discovered.

3.3.16 Parameter Confidence Intervals

Least squares methods ([nl2sol](#), [nlssol_sqp](#), [optpp_g_newton](#)) compute confidence intervals on the calibration parameters.

3.3.17 Best Model Responses (without configuration variables)

When performing calibration with experimental data (but no configuration variables), Dakota records, in addition to the best residuals, the best original model responses.

3.3.18 Best Model Responses (with configuration variables)

When performing calibration with experimental data that includes configuration variables, Dakota reports the best model responses for each experiment. These results include the configuration variables, stored in the scheme described in the [variables](#) section, and the model responses.

3.3.19 Multistart and Pareto Set

The [multi_start](#) and [pareto_set](#) methods are meta-iterators that control multiple optimization sub-iterators. For both methods, Dakota stores the results of the sub-iterators (best parameters and best results). For [multi_start](#), Dakota additionally stores the initial points, and for [pareto_set](#), it stores the objective function weights.

3.4 Organization of Evaluations

An *evaluation* is a mapping from variables to responses performed by a Dakota model or interface. Beginning with release 6.10, Dakota has the ability to report evaluation history in HDF5 format. The HDF5 format offers many advantages over existing console output and [tabular output](#). Requiring no "scraping", it is more convenient for most users than the former, and being unrestricted to a two-dimensional, tabular arrangement of information, it is far richer than the latter.

This section begins by describing the Dakota components that can generate evaluation data. It then documents the high-level organization of the data from those components. Detailed documentation of the individual datasets (the "low-level" organization) where data are stored follows. Finally, information is provided concerning input keywords that control which components report evaluations.

3.4.1 Sources of Evaluation Data

Evaluation data are produced by only two kinds of components in Dakota: **models** and **interfaces**. The purpose of this subsection is to provide a basic description of models and interfaces for the purpose of equipping users to manage and understand HDF5-format evaluation data.

Because interfaces and models must be specified in even simple Dakota studies, most novice users of Dakota will have some familiarity with these concepts. However, the exact nature of the relationship between methods, models, and interfaces may be unclear. Moreover, the models and interfaces present in a Dakota study are not always limited to those specified by the user. Some input keywords or combinations of components cause Dakota to create new models or interfaces "behind the scenes" and without the user's direct knowledge. Not only can user-specified models and interfaces write evaluation data to HDF5, but also these auto-generated components. Accordingly, it

may be helpful for consumers of Dakota's evaluation data to have a basic understanding of how Dakota creates and employs models and interfaces.

Consider first the input file shown here.

```
environment
  tabular_data
  results_output
  hdf5

method
  id_method 'sampling'
  sampling
    samples 20
  model_pointer 'sim'

model
  id_model 'sim'
  single
  interface_pointer 'tb'

variables
  uniform_uncertain 2
  descriptors 'x1' 'x2'
  lower_bounds 0.0 0.0
  upper_bounds 1.0 1.0

responses
  response_functions 1
  descriptors 'f'
  no_gradients
  no_hessians

interface
  id_interface 'tb'
  fork
  analysis_drivers 'text_book'
```

This simple input file specifies a single method of type [sampling](#), which also has the Id 'sampling'. The 'sampling' method possesses a [model](#) of type [single](#) (alias simulation) named 'sim', which it uses to perform evaluations. (Dakota would have automatically generated a single model had one not been specified.) That is to say, for each variables-to-response mapping required by the method, it provides variables to the model and receives back responses from it.

Single/simulation models like 'sim' perform evaluations by means of an interface, typically an interface to an external simulation. In this case, the interface is 'tb'. The model passes the variables to 'tb', which executes the `text_book` driver, and receives back responses.

It is clear that two components produce evaluation data in this study. The first is the single model 'sim', which receives and fulfills evaluation requests from the method 'sampling', and the second is the interface 'tb', which similarly receives requests from 'sim' and fulfills them by running the `text_book` driver.

Because [tabular data](#) was requested in the environment block, a record of the model's evaluations will be reported to a tabular file. The interface's evaluations could be dumped from the restart file using `dakota_restart_util`.

If we compared these evaluation histories from 'sim' and 'tb', we would see that they are identical to one another. The model 'sim' is a mere "middle man" whose only responsibility is passing variables from the method down to the interface, executing the interface, and passing responses back up to the method. However, this is not always the case.

For example, if this study were converted to a gradient-based optimization using [optpp_q_newton](#), and the user specified [numerical_gradients](#) :

```
# model and interface same as above. Replace the method, variables, and responses with:

method
  id_method 'opt'
  optpp_q_newton

variables
```

```

continuous_design 2
  descriptors 'x1' 'x2'
  lower_bounds 0.0 0.0
  upper_bounds 1.0 1.0

responses
  objective_functions 1
  descriptors 'f'
  numerical_gradients
  no_hessians

```

Then the model would have the responsibility of performing finite differencing to estimate gradients of the response 'f' requested by the method. Multiple function evaluations of 'tb' would map to a single gradient evaluation at the model level, and the evaluation histories of 'sim' and 'tb' would contain different information.

Note that because it is unwieldy to report gradients (or Hessians) in a tabular format, they are not written to the tabular file, and historically were available only in the console output. The HDF5 format provides convenient access to both the "raw" evaluations performed by the interface and higher level model evaluations that include estimated gradients.

This pair of examples hopefully provides a basic understanding of the flow of evaluation data between a method, model, and interface, and explains why models and interfaces are producers of evaluation data.

Next consider a somewhat more complex study that includes a Dakota model of type [surrogate](#). A surrogate model performs evaluations requested by a method by executing a special kind of interface called an *approximation interface*, which Dakota implicitly creates without the direct knowledge of the user. Approximation interfaces are a generic container for the various kinds of surrogates Dakota can use, such as [gaussian processes](#).

A Dakota model of type global surrogate may use a user-specified [dace method](#) to construct the actual underlying model(s) that it evaluates via its approximation interface. The dace method will have its own model (typically of type single/simulation), which will have a user-specified interface.

In this more complicated case there are at least four components that produce evaluation data: (1) the surrogate model and (2) its approximation interface, and (3) the dace method's model and (4) its interface. Although only components (1), (3), and (4) are user-specified, evaluation data produced by (2) may be written to HDF5, as well. (As [explained below](#), only evaluations performed by the surrogate model and the dace interface will be recorded by default. This can be overridden using [hdf5](#) sub-keywords.) This is an example where "extra" and potentially confusing data appears in Dakota's output due to an auto-generated component.

An important family of implicitly-created models is the *recast* models, which have the responsibility of transforming variables and responses. One type of recast called a *data transform model* is responsible for computing residuals when a user provides [experimental data](#) in a calibration study. *Scaling* recast models are employed when scaling is requested by the user for variables and/or responses.

Recast models work on the principle of function composition, and "wrap" a submodel, which may itself also be a recast model. The innermost model in the recursion often will be the simulation or surrogate model specified by the user in the input file. Dakota is capable of recording evaluation data at each level of recast.

3.4.2 High-level Organization of Evaluation Data

This subsection describes how evaluation data produced by models and interfaces are organized at high level. A detailed description of the datasets and subgroups that contain evaluation data for a specific model or interface is given in the [next subsection](#).

Two top level groups contain evaluation data, `/interfaces` and `/models`.

3.4.2.1 Interfaces

Because interfaces can be executed by more than one model, interface evaluations are more precisely thought of as evaluations of an interface/model combination. Consequently, interface evaluations are grouped not only by interface Id ('tb' in the example above), but also the Id of the model that requested them ('sim').

```
/interfaces/<interface Id>/<model Id>/
```

If the user does not provide an Id for an interface that he specifies, Dakota assigns it the Id `NO_ID`. Approximation interfaces receive the Id `APPROX_INTERFACE_<N>`, where N is an incrementing integer beginning at 1. Other kinds of automatically generated interfaces are named `NOSPEC_INTERFACE_ID_<N>`.

3.4.2.2 Models

The top-level group for model evaluations is `/models`. Within this group, model evaluations are grouped by type: `simulation`, `surrogate`, `nested`, or `recast`, and then by model Id. That is:

```
/models/<type>/<model Id>/
```

Similar to interfaces, user-specified models that lack an Id are given one by Dakota. A single model is named `NO_MODEL_ID`. Some automatically generated models receive the name `NOSPEC_MODEL_ID`.

Recast models are a special case and receive the name `RECAST_<WRAPPED-MODEL>_<TYPE>_<N>`. In this string:

- `WRAPPED-MODEL` is the Id of the innermost wrapped model, typically a user-specified model
- `TYPE` is the specific kind of recast. The three most common recasts are:
 - `RECAST`: several generic responsibilities, including summing objective functions to present to a single-objective optimizer
 - `DATA_TRANSFORM`: Compute residuals in a calibration
 - `SCALING`: scale variables and responses
- N is an incrementing integer that begins with 1. It is employed to distinguish recasts of the same type that wrap the same underlying model.

The model's evaluations may be the result of combining information from multiple sources. A simulation/single model will receive all the information it requires from its interface, but more complicated model types may use information not only from interfaces, but also other models and the results of method executions. Nested models, for instance, receive information from a submethod (the mean of a response from a sampling study, for instance) and potentially also an [optional interface](#).

The sources of a model's evaluations may be roughly identified by examining the contents of that model's `sources` group. The `sources` group contains softlinks (note: softlinks are an HDF5 feature analogous to soft or symbolic links on many file systems) to groups for the interfaces, models, or methods that the model used to produce its evaluation data. (At this time, Dakota does not report the specific interface or model evaluations or method executions that were used to produce a specific model evaluation, but this is a planned feature.)

Method results likewise have a `sources` group that identifies the models or methods employed by that method. By following the softlinks contained in a method's or model's `sources` group, it is possible to "drill down" from a method to its ultimate sources of information. In the sampling example above, interface evaluations performed via the 'sim' model at the request of the 'sampling' method could be obtained at the HDF5 path: `/methods/sampling/sources/sim/sources/tb/`

3.4.3 Low-Level Organization of Evaluation Data

Within each model and interface's "high-level" group, evaluation data are stored according to a "low-level" schema. This section describes the "low-level" schema.

Data are divided first of all into `variables`, `responses`, and `properties` groups. In addition, if a user specifies metadata responses in his Dakota input, a `metadata` dataset will be present.

3.4.3.1 Variables

The `variables` group contains datasets that store the variables information for each evaluation. Four datasets may be present, one for each "domain": `continuous`, `discrete_integer`, `discrete_string`, and

`discrete_real`. These datasets are two-dimensional, with a row (0th dimension) for each evaluation and a column (1st dimension) for each variable. The 0th dimension has one dimension scale for the integer-valued evaluation Id. The 1st dimension has two scales. The 0th scale contains descriptors of the variables, and the 1st contains their variable Ids. In this context, the Ids are a 1-to-N ranking of the variables in Dakota "input spec" order.

3.4.3.2 Responses

The responses group contains datasets for functions and, when available, gradients and Hessians.

Functions: The `functions` dataset is two-dimensional and contains function values for all responses. Like the variables datasets, evaluations are stored along the 0th dimension, and responses are stored along the 1st. The evaluation Ids and response descriptors are attached as scales to these axes, respectively.

Gradients: The `gradients` dataset is three-dimensional. It has the shape $evaluations \times responses \times variables$. Dakota supports a specification of `mixed_gradients`, and the `gradients` dataset is sized and organized such that only those responses for which gradients are available are stored. When `mixed_gradients` are employed, a response will not necessarily have the same index in the `functions` and `gradients` datasets.

Because it is possible that the gradient could be computed with respect to any of the continuous variables, active or inactive, that belong to the associated model, the `gradients` dataset is sized to accommodate gradients taken with respect to all continuous variables. Components that were not included in a particular evaluation will be set to NaN (not a number), and the `derivative_variables_vector` (in the metadata group) for that evaluation can be examined as well.

Hessians: Hessians are stored in a four-dimensional dataset, $evaluations \times responses \times variables \times variables$. The `hessians` dataset shares many of the characteristics with the `gradients`: in the `mixed_hessians` case, it will be smaller in the response dimension than the `functions` dataset, and unrequested components are set to NaN.

3.4.3.3 Properties

The `properties` group contains up to four members.

Active Set Vector: The first is the `active_set_vector` dataset. It is two dimensional, with rows corresponding to evaluations and columns corresponding to responses. Each element contains an integer in the range 0-7, which indicates the request (function, gradient, Hessian) for the corresponding response for that evaluation. The 0th dimension has the evaluations Ids scale, and the 1st dimension has two scales: the response descriptors and the "default" or "maximal" ASV, an integer 0-7 for each response that indicates the information (function, gradient, Hessian) that possibly could have been requested during the study.

Derivative Variables Vector: The second item in the properties group is the `derivative_variables_vector` dataset. It is included only when gradients or Hessians are available. Like the ASV, it is two-dimensional. Each column of the DVV dataset corresponds to a continuous variable and contains a 0 or 1, indicating whether gradients and Hessians were computed with respect to that variable for the evaluation. The 0th dimension has the evaluation Ids as a scale, and the 1st dimension has two scales. The 0th is the descriptors of the continuous variables. The 1st contains the variable Ids of the continuous variables.

Analysis Components: The third member of the properties group is the `analysis_components` dataset. It is a 1D dataset that is present only when the user specified analysis components, and it contains those components as strings.

The final possible member of the properties group is the `variable_parameters` group. It is included only for models, which possess variables, and is described in a separate section below.

3.4.3.4 Metadata

Beginning with release 6.16, Dakota supports response `metadata`. If configured, metadata values are stored in the `metadata` dataset.

3.4.4 Selecting Models and Interfaces to Store

When HDF5 output is enabled (by including the `hdf5` keyword), then by default evaluation data for the following components will be stored:

- The model that belongs to the top-level method. (Currently, if the top-level method is a metaiterator such as `method-hybrid`, no model evaluation data will be stored.)
- All simulation interfaces. (interfaces of type `fork`, `system`, `direct`, etc).

The user can override these defaults using the keywords `model_selection` and `interface_selection`.

The choices for `model_selection` are:

- `top_method` : (*default*) Store evaluation data for the top method's model only.
- `all_methods` : Store evaluation data for all models that belong directly to a method. Note that a these models may be recasts of user-specified models, not the user-specified models themselves.
- `all` : Store evaluation data for all models.
- `none` : Store evaluation data for no models.

The choices for `interface_selection` are:

- `simulation` : (*default*) Store evaluation data for simulation interfaces.
- `all` : Store evaluation data for all interfaces.
- `none` : Store evaluation data for no interfaces.

If a model or interface is excluded from storage by these selections, then they cannot appear in the `sources` group for methods or models.

3.5 Distribution Parameters

Variables are characterized by parameters such as the mean and standard deviation or lower and upper bounds. Typically, users provide these parameters as part of their input to Dakota, but Dakota itself may also compute them as it scales and transforms variables, normalizes empirical distributions (e.g. for `histogram_bin_uncertain` variables), or calculates alternative parameterizations (lambda and zeta vs mean and standard deviation for a `lognormal_uncertain`).

Beginning with release 6.11, models write their variable's parameters to HDF5. The information is located in each model's `properties/variable_parameters` subgroup. Within this group, parameters are stored by Dakota variable type (e.g. `normal_uncertain`), with one 1D dataset per type. The datasets have the same names as their variable types and have one element per variable. Parameters are stored by name.

Consider the following variable specification, which includes two normal and two uniform variables:

```
variables
  normal_uncertain 2
    descriptors 'nuv_1' 'nuv_2'
    means 0.0 1.0
    std_deviations 1.0 0.5
  uniform_uncertain 2
    descriptors 'uuv_1' 'uuv_2'
    lower_bounds -1.0 0.0
    upper_bounds 1.0 1.0
```

Given this specification, and assuming a model ID of "tb_model", Dakota will write two 1D datasets, both of length 2, to the group `/models/simulation/tb_model/metadata/variable_parameters`, the first named `normal_uncertain`, and the second named `uniform_uncertain`. Using a JSON-like representation for illustration, the `normal_uncertain` dataset will appear as:

```
[
  {
    "mean": 0.0,
    "std_deviation": 1.0,
    "lower_bound": -inf,
    "upper_bound": inf
  },
  {
    "mean": 1.0,
    "std_deviation": 0.5,
    "lower_bound": -inf,
    "upper_bound": inf
  }
]
```

The `uniform_uncertain` dataset will contain:

```
[
  {
    "lower_bound": -1.0,
    "upper_bound": 1.0
  },
  {
    "lower_bound": 0.0,
    "upper_bound": 1.0
  }
]
```

In these representations of the `normal_uncertain` and `uniform_uncertain` datasets, the outer square brackets ([]) enclose the dataset, and each element within the datasets are enclosed in curly braces ({}). The curly braces are meant to indicate that the elements are dictionary-like objects that support access by string field name. A bit more concretely, the following code snippet demonstrates reading the mean of the second normal variable, `nuv_2`.

```
1 import h5py
2
3 with h5py.File("dakota_results.h5") as h:
4     model = h["/models/simulation/tb_model/"]
5     # nu_vars is the dataset that contains distribution parameters for
6     # normal_uncertain variables
7     nu_vars = model["variable_parameters/normal_uncertain"]
8     nuv_2_mu = nu_vars[1]["mean"] # 1 is the 0-based index of nuv_2, and
9                                   # "mean" is the name of the field where
10                                  # the mean is stored; nuv_2_mu now contains
11                                  # 1.0.
```

The feature in HDF5 that underlies this name-based storage of fields is compound datatypes, which are similar to C/C++ structs or Python dictionaries. Further information about how to work with compound datatypes is available in the `h5py` documentation.

3.5.1 Naming Conventions and Layout

In most cases, datasets for storing parameters have names that match their variable types. The `normal_uncertain` and `uniform_uncertain` datasets illustrated above are examples. Exceptions include types such as `discrete_design_set`, which has string, integer, and real subtypes. For these, the dataset name is the top-level type with `_string`, `_int`, or `_real` appended: `discrete_design_set_string`, `discrete_design_set_int`, and `discrete_design_set_real`.

Most Dakota variable types have scalar parameters. For these, the names of the parameters are generally the singular form of the associated Dakota keyword. For example, `triangular_uncertain` variables are characterized in Dakota input using the plural keywords `modes`, `lower_bounds`, and `upper_bounds`. The singular field names are, respectively, "mode", "lower_bound", and "upper_bound". In this case, all three parameters are real-valued and stored as floating point numbers, but variable types/fields can also be integer-valued (e.g. `binomial_uncertain/num_trials`) or string-valued.

Some variable/parameter fields contain 1D arrays or vectors of information. Consider `histogram_bin_uncertain` variables, for which the user specifies not just one value, but an ordered collection of abscissas and corresponding

ordinates or counts. Dakota stores the abscissas in the "abscissas" field, which is a 1D dataset of floating-point numbers. It similarly stores the counts in the "counts" field. (In this case, only the normalized counts are stored, regardless of whether the user provided counts or ordinates.)

When the user specifies more than one `histogram_bin_uncertain` variable, it often is also necessary to include the `pairs_per_variable` keyword to divide the `abscissa/count` pairs among the variables. This raises the question of how lists of parameters that vary in length across the variables ought to be stored.

Although HDF5 supports variable-length datasets, for simplicity (and due to limitations in `h5py` at the time of the 6.11 release), Dakota stores vector parameter fields in conventional fixed-length datasets. The lengths of these datasets are determined at runtime in the following way: For a particular variable type and field, the field for all variables is sized to be large enough to accommodate the variable with the longest list of parameters. Any unused space for a particular variable is filled with `NaN` (if the parameter is real-valued), `INTMAX` (integer-valued), or an empty string (`string`-valued). In addition, each variable has an additional field, "num_elements", that reports the number of elements in the fields that contain actual data and not fill values.

Consider this example, in which the user has specified a pair of `histogram_bin_uncertain` variables. The first has 3 pairs, and the second has 4.

```
variables
  histogram_bin_uncertain 2
    pairs_per_variable 2 3
    abscissas 0.0 0.5 1.0
              -1.0 -0.5 0.5 1.0
    counts    0.25 0.75 0.0
              0.2 0.4 0.2 0.0
```

For this specification, Dakota will write a dataset named `histogram_bin_uncertain` to the `metadata/variable-parameters/` subgroup for the model. It will be of length 2, one element for each variable, and contain the following:

```
[
  {
    "num_elements": 3,
    "abscissas": [0.0, 0.5, 1.0, NaN],
    "counts": [0.25, 0.75, 0.0, NaN]
  },
  {
    "num_elements": 4,
    "abscissas": [-1.0, -0.5, 0.5, 1.0],
    "counts": [0.2, 0.4, 0.2, 0.0]
  }
]
```

3.5.2 h5py Examples

The fields available for a variable parameters dataset can be determined in `h5py` by examining the datatype of the dataset.

```
1 import h5py
2 with h5py.File("dakota_results.h5") as h:
3     model = h["/models/simulation/NO_MODEL_ID/"]
4     md = model["metadata/variable_parameters"]
5     nu = md["normal_uncertain"]
6     nu_param_names = nu.dtype.names
7     # nu_param_names is a tuple of strings: ('mean', 'std_deviation',
8     # 'lower_bound', 'upper_bound')
```

3.5.3 Known Limitations

`h5py` has a known bug that prevents parameters for some types of variables from being accessed (the Python interpreter crashes with a segfault). These include:

- `histogram_point_uncertain` string
- `discrete_uncertain_set` string

3.5.4 Metadata

The variable parameter datasets have two dimension scales. The first (index 0) contains the variable descriptors, and the second (index 1) contains variable ids. Available Parameters

3.5.5 Parameter Listing for All Types

The table below lists all Dakota variables and parameters that can be stored.

Chapter 4

Test Problems

This page contains additional information about two test problems that are used in Dakota examples throughout the Dakota manuals [Textbook](#) and [Rosenbrock](#).

Many of these examples are also used as code verification tests. The examples are run periodically and the results are checked against known solutions. This ensures that the algorithms are correctly implemented.

Additional test problems are described in the User's Manual.

4.1 Textbook

The two-variable version of the "textbook" test problem provides a nonlinearly constrained optimization test case. It is formulated as:

$$\begin{aligned} & \text{minimize } f = (x_1 - 1)^4 + (x_2 - 1)^4 \\ & \text{subject to } g_1 = x_1^2 - \frac{x_2}{2} \leq 0 \\ & \quad \quad \quad g_2 = x_2^2 - \frac{x_1}{2} \leq 0 \\ & \quad \quad \quad 0.5 \leq x_1 \leq 5.8 \\ & \quad \quad \quad -2.9 \leq x_2 \leq 2.9 \end{aligned} \quad (\text{textbookform})$$

Contours of this test problem are illustrated in the next two figures.

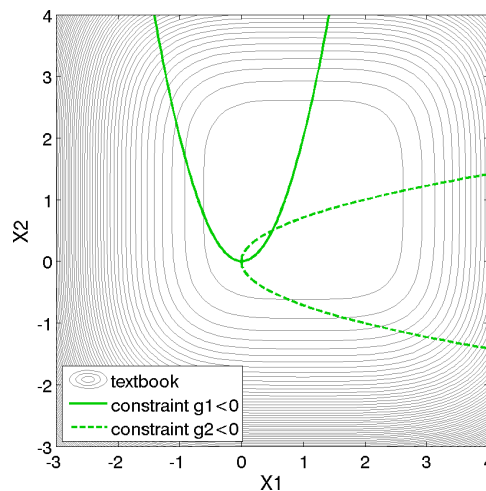


Figure 4.1: Contours of the textbook problem on the $[-3,4] \times [-3,4]$ domain. The feasible region lies at the intersection of the two constraints g_1 (solid) and g_2 (dashed).

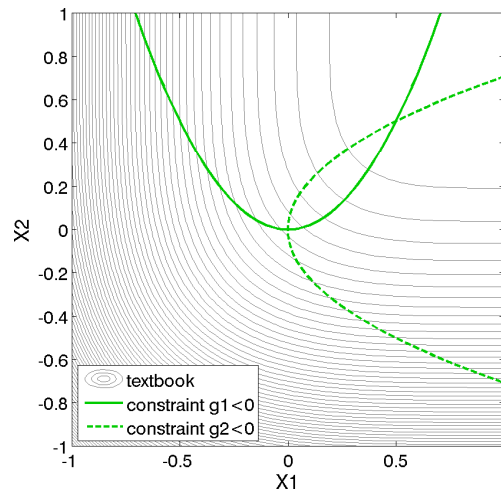


Figure 4.2: Contours of the textbook problem zoomed into an area containing the constrained optimum point $(x_1, x_2) = (0.5, 0.5)$. The feasible region lies at the intersection of the two constraints g_1 (solid) and g_2 (dashed).

For the textbook test problem, the unconstrained minimum occurs at $(x_1, x_2) = (1, 1)$. However, the inclusion of the constraints moves the minimum to $(x_1, x_2) = (0.5, 0.5)$. Equation [textbookform](#) presents the 2-dimensional form of the textbook problem. An extended formulation is stated as

$$\begin{aligned}
 & \text{minimize } f = \sum_{i=1}^n (x_i - 1)^4 \\
 & \text{subject to } g_1 = x_1^2 - \frac{x_2}{2} \leq 0 \quad (\text{tbe}) \\
 & \quad \quad \quad g_2 = x_2^2 - \frac{x_1}{2} \leq 0 \\
 & \quad \quad \quad 0.5 \leq x_1 \leq 5.8 \\
 & \quad \quad \quad -2.9 \leq x_2 \leq 2.9
 \end{aligned}$$

where n is the number of design variables. The objective function is designed to accommodate an arbitrary number of design variables in order to allow flexible testing of a variety of data sets. Contour plots for the $n = 2$ case have been shown previously.

For the optimization problem given in Equation [tbe](#), the unconstrained solution

(`num_nonlinear_inequality_constraints` set to zero) for two design variables is:

$$\begin{aligned}
 x_1 &= 1.0 \\
 x_2 &= 1.0
 \end{aligned}$$

with

$$f^* = 0.0$$

The solution for the optimization problem constrained by g_1 (`num_nonlinear_inequality_constraints` set to one) is:

$$\begin{aligned}
 x_1 &= 0.763 \\
 x_2 &= 1.16
 \end{aligned}$$

with

$$\begin{aligned}
 f^* &= 0.00388 \\
 g_1^* &= 0.0 \text{ (active)}
 \end{aligned}$$

The solution for the optimization problem constrained by g_1 and g_2 (num_nonlinear_inequality_constraints set to two) is:

$$\begin{aligned}x_1 &= 0.500 \\x_2 &= 0.500\end{aligned}$$

with

$$\begin{aligned}f^* &= 0.125 \\g_1^* &= 0.0 \text{ (active)} \\g_2^* &= 0.0 \text{ (active)}\end{aligned}$$

Note that as constraints are added, the design freedom is restricted (the additional constraints are active at the solution) and an increase in the optimal objective function is observed.

4.2 Rosenbrock

The Rosenbrock function [34] is a well-known test problem for optimization algorithms. The standard formulation includes two design variables, and computes a single objective function. This problem can also be posed as a least-squares optimization problem with two residuals to be minimized because the objective function is the sum of squared terms.

Standard Formulation

The standard two-dimensional formulation can be stated as

$$\text{minimize } f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (\text{rosenstd})$$

Surface and contour plots for this function are shown in the Dakota User's Manual.

The optimal solution is:

$$\begin{aligned}x_1 &= 1.0 \\x_2 &= 1.0\end{aligned}$$

with

$$f^* = 0.0$$

A Least-Squares Optimization Formulation

This test problem may also be used to exercise least-squares solution methods by recasting the standard problem formulation into:

$$\text{minimize } f = (f_1)^2 + (f_2)^2 \quad (\text{rosenls})$$

where

$$f_1 = 10(x_2 - x_1^2) \quad (\text{rosenr1})$$

and

$$f_2 = 1 - x_1 \quad (\text{rosenr2})$$

are residual terms.

The included analysis driver can handle both formulations. In the `dakota/share/dakota/test` directory, the `rosenbrock` executable (compiled from `Dakota_Source/test/rosenbrock.cpp`) checks the number of response functions passed in the parameters file and returns either an objective function (as computed from Equation [rosenstd](#)) for use with optimization methods or two least squares terms (as computed from Equations [rosenr1](#) - [rosenr2](#)) for use with least squares methods. Both cases support analytic gradients of the function set with respect to the design variables. See the User's Manual for examples of both cases (search for Rosenbrock).

Chapter 5

Dakota Input Specification

Dakota input is specified in a text file, e.g., `dakota_uq.in` containing blocks of keywords that control program behavior. This section describes the format and admissible elements of an input file.

5.1 Dakota Keywords

Valid Dakota input keywords are dictated by `dakota.xml`, included in source and binary distributions of Dakota. This specification file is used with the NIDR [30] parser to validate user input and is therefore the definitive source for input syntax, capability options, and optional and required capability sub-parameters for any given Dakota version. A more readable variant of the specification `dakota.input.summary` is also distributed.

While complete, users may find `dakota.input.summary` overwhelming or confusing and will likely derive more benefit from adapting example input files to a particular problem. Some examples can be found here: [Sample Input Files](#). Advanced users can master the many input specification possibilities by understanding the structure of the input specification file.

5.2 Input Spec Overview

Refer to the `dakota.input.summary` file, in [Input Spec Summary](#), for all current valid input keywords.

- The summary describes every keyword including:
 - Whether it is required or optional
 - Whether it takes `ARGUMENTS` (always required) Additional notes about `ARGUMENTS` can be found here: [Specifying Arguments](#).
 - Whether it has an `ALIAS`, or synonym
 - Which additional keywords can be specified to change its behavior
- Additional details and descriptions are described in [Keywords Area](#)
- For additional details on NIDR specification logic and rules, refer to [30] (Gay, 2008).

5.2.1 Common Specification Mistakes

Spelling mistakes and omission of required parameters are the most common errors. Some causes of errors are more obscure:

- Documentation of new capability sometimes lags its availability in source and executables, especially stable releases. When parsing errors occur that the documentation cannot explain, reference to the particular input

specification `dakota.input.summary` used in building the executable, which is installed alongside the executable, will often resolve the errors.

- If you want to compare results with those obtained using an earlier version of Dakota (prior to 4.1), your input file for the earlier version must use backslashes to indicate continuation lines for Dakota keywords. For example, rather than

```
# Comment about the following "responses" keyword...
responses,
  objective_functions = 1
  # Comment within keyword "responses"
  analytic_gradients
# Another comment within keyword "responses"
no_hessians
```

you would need to write

```
# Comment about the following "responses" keyword...
responses,
  objective_functions = 1
  # Comment within keyword "responses"
  analytic_gradients
# Another comment within keyword "responses"
no_hessians
```

with no white space (blanks or tabs) after the `\` character.

In most cases, the Dakota parser provides error messages that help the user isolate errors in input files. Running `dakota -input dakota_study.in -check` will validate the input file without running the study.

5.2.2 Specifying Arguments

Some keywords, such as those providing bounds on variables, have an associated list of values or strings, referred to as arguments.

When the same value should be repeated several times in a row, you can use the notation `N*value` instead of repeating the value `N` times.

For example

```
lower_bounds -2.0 -2.0 -2.0
upper_bounds 2.0 2.0 2.0
```

could also be written

```
lower_bounds 3*-2.0
upper_bounds 3* 2.0
```

(with optional spaces around the `*`).

Another possible abbreviation is for sequences: `L:S:U` (with optional spaces around the `:`) is expanded to `L L+S L+2*S ... U`, and `L:U` (with no second colon) is treated as `L:1:U`.

For example, in one of the test examples distributed with Dakota (test case 2 of `test/dakota_uq_textbook-_sop_lhs.in`),

```
histogram_point = 2
abscissas = 50. 60. 70. 80. 90.
           30. 40. 50. 60. 70.
counts = 10 20 30 20 10
         10 20 30 20 10
```

could also be written

```
histogram_point = 2
abscissas = 50 : 10 : 90
           30 : 10 : 70
counts = 10:10:30 20 10
         10:10:30 20 10
```


Count and sequence abbreviations can be used together. For example

```
response_levels =
  0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
  0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

can be abbreviated

```
response_levels =
  2*0.0:0.1:1.0
```

5.3 Sample Input Files

A Dakota input file is a collection of fields from the `dakota.input.summary` file that describe the problem to be solved by Dakota. Several brief examples follow below.

Dakota 6.14 and newer include a refreshed Examples Library, ranging from examples of Dakota input files and basic studies through complete case studies. These can be found in binary distributions in `dakota/share/dakota/examples/official` (`dakota/dakota-examples` in source distributions). Legacy examples, including many from the Dakota software manuals appear in other directories in `dakota/share/dakota/examples` (`dakota/examples` for source).

Sample 1: Optimization

The following sample input file shows single-method optimization of the Textbook Example (see [Textbook](#)) using DOT's modified method of feasible directions. A similar file is available as `dakota/share/dakota/examples/users/textbook_opt_conmin.in`.

```
# Dakota Input File: textbook_opt_conmin.in
environment
  tabular_data
    tabular_data_file = 'textbook_opt_conmin.dat'

method
# dot_mmfd #DOT performs better but may not be available
  conmin_mfd
  max_iterations = 50
  convergence_tolerance = 1e-4

variables
  continuous_design = 2
  initial_point 0.9 1.1
  upper_bounds 5.8 2.9
  lower_bounds 0.5 -2.9
  descriptors 'x1' 'x2'

interface
  direct
  analysis_driver = 'text_book'

responses
  objective_functions = 1
  nonlinear_inequality_constraints = 2
  numerical_gradients
  method_source dakota
  interval_type central
  fd_gradient_step_size = 1.e-4
  no_hessians
```

Sample 2: Least Squares (Calibration)

The following sample input file shows a nonlinear least squares (calibration) solution of the Rosenbrock Example (see [Rosenbrock](#)) using the NL2SOL method. A similar file is available as `dakota/share/dakota/examples/users/rosenbrock_opt_nls.in`

```
# Dakota Input File: rosen_opt_nls.in
```

```

environment
  tabular_data
    tabular_data_file = 'rosen_opt_nls.dat'

method
  max_iterations = 100
  convergence_tolerance = 1e-4
  nl2sol

model
  single

variables
  continuous_design = 2
  initial_point   -1.2  1.0
  lower_bounds    -2.0  -2.0
  upper_bounds    2.0   2.0
  descriptors     'x1'  "x2"

interface
  analysis_driver = 'rosenbrock'
  direct

responses
  calibration_terms = 2
  analytic_gradients
  no_hessians

```

Sample 3: Nondeterministic Analysis

The following sample input file shows Latin Hypercube Monte Carlo sampling using the Textbook Example (see [Textbook](#)). A similar file is available as `dakota/share/dakota/test/dakota_uq_textbook_lhs.in`.

```

method,
  sampling,
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    sample_type lhs

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds    = 199.3, 474.63
  upper_bounds    = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs       = 3 4
  abscissas       = 5 8 10 .1 .2 .3 .4
  counts          = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain = 1
  num_pairs       = 2
  abscissas       = 3 4
  counts          = 1 1
  descriptors      = 'TF3h'

interface,
  fork_async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3

```

```
no_gradients
no_hessians
```

Sample 4: Parameter Study

The following sample input file shows a 1-D vector parameter study using the Textbook Example (see [Textbook](#)). It makes use of the default environment and model specifications, so they can be omitted. A similar file is available in the test directory as `dakota/share/dakota/examples/users/rosen_ps_vector.in`.

```
# Dakota Input File: rosen_ps_vector.in
environment
  tabular_data
  tabular_data_file = 'rosen_ps_vector.dat'

method
  vector_parameter_study
  final_point = 1.1 1.3
  num_steps = 10

variables
  continuous_design = 2
  initial_point -0.3 0.2
  descriptors 'x1' "x2"

interface
  analysis_driver = 'rosenbrock'
  direct

responses
  objective_functions = 1
  no_gradients
  no_hessians
```

Sample 5: Hybrid Strategy

The following sample input file shows a hybrid environment using three methods. It employs a genetic algorithm, pattern search, and full Newton gradient-based optimization in succession to solve the Textbook Example (see [Textbook](#)). A similar file is available as `dakota/share/dakota/examples/users/textbook_hybrid_strat.in`.

```
environment
  hybrid sequential
  method_list = 'PS' 'PS2' 'NLP'

method
  id_method = 'PS'
  model_pointer = 'M1'
  coliny_pattern_search stochastic
  seed = 1234
  initial_delta = 0.1
  variable_tolerance = 1.e-4
  solution_accuracy = 1.e-10
  exploratory_moves basic_pattern
  #verbose output

method
  id_method = 'PS2'
  model_pointer = 'M1'
  max_function_evaluations = 10
  coliny_pattern_search stochastic
  seed = 1234
  initial_delta = 0.1
  variable_tolerance = 1.e-4
  solution_accuracy = 1.e-10
  exploratory_moves basic_pattern
  #verbose output

method
  id_method = 'NLP'
  model_pointer = 'M2'
```

```

    optpp_newton
    gradient_tolerance = 1.e-12
    convergence_tolerance = 1.e-15
    #verbose output

model
  id_model = 'M1'
  single
  variables_pointer = 'V1'
  interface_pointer = 'I1'
  responses_pointer = 'R1'

model
  id_model = 'M2'
  single
  variables_pointer = 'V1'
  interface_pointer = 'I1'
  responses_pointer = 'R2'

variables
  id_variables = 'V1'
  continuous_design = 2
  initial_point 0.6 0.7
  upper_bounds 5.8 2.9
  lower_bounds 0.5 -2.9
  descriptors 'x1' 'x2'

interface
  id_interface = 'I1'
  direct
  analysis_driver= 'text_book'

responses
  id_responses = 'R1'
  objective_functions = 1
  no_gradients
  no_hessians

responses
  id_responses = 'R2'
  objective_functions = 1
  analytic_gradients
  analytic_hessians

```

Additional example input files, as well as the corresponding output, are provided in the Tutorial chapter of the Users Manual [5].

5.4 Input Spec Summary

This file is derived automatically from `dakota.xml`, which is used in the generation of parser system files that are compiled into the Dakota executable. Therefore, these files are the definitive source for input syntax, capability options, and associated data inputs. Refer to the Developers Manual information on how to modify the input specification and propagate the changes through the parsing system.

Key features of the input specification and the associated user input files include:

- In the input specification, required individual specifications simply appear, optional individual and group specifications are enclosed in `[]`, required group specifications are enclosed in `()`, and either-or relationships are denoted by the `|` symbol. These symbols only appear in `dakota.input.summary`; they must not appear in actual user input files.
- **Keyword specifications** (i.e., `environment`, `method`, `model`, `variables`, `interface`, and `responses`) begin with the keyword possibly preceded by white space (blanks, tabs, and newlines) both in the input specifications and in user input files. For readability, keyword specifications may be spread across several lines. Earlier versions of Dakota (prior to 4.1) required a backslash character (`\`) at the ends of intermediate lines of a keyword. While such backslashes are still accepted, they are no longer required.

- Some of the keyword components within the input specification indicate that the user must supply `INTEGER`, `REAL`, `STRING`, `INTEGERLIST`, `REALLIST`, or `STRINGLIST` data as part of the specification. In a user input file, the "=" is optional, data in a `LIST` can be separated by commas or whitespace, and the `STRING` data are enclosed in single or double quotes (e.g., `'text_book'` or `"text_book"`).
- In user input files, input is largely order-independent (except for entries in lists of data), case insensitive, and white-space insensitive. Although the order of input shown in the [Sample Input Files](#) generally follows the order of options in the input specification, this is not required.
- In user input files, specifications may be abbreviated so long as the abbreviation is unique. For example, the `npsol_sqp` specification within the method keyword could be abbreviated as `npsol`, but `dot_sqp` should not be abbreviated as `dot` since this would be ambiguous with other `DOT` method specifications.
- In both the input specification and user input files, comments are preceded by `#`.
- `ALIAS` refers to synonymous keywords, which often exist for backwards compatibility. Users are encouraged to use the most current keyword.

dakota.input.summary:

```

KEYWORD01 environment
  [ tabular_data ALIAS tabular_graphics_data
    [ tabular_data_file ALIAS tabular_graphics_file STRING ]
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
    | annotated
    | freeform ]
  ]
  [ output_file STRING ]
  [ error_file STRING ]
  [ read_restart STRING
    [ stop_restart INTEGER >= 0 ]
  ]
  [ write_restart STRING ]
  [ output_precision INTEGER >= 0 ]
  [ results_output
    [ results_output_file STRING ]
    [ text ]
    [ hdf5
      [ model_selection
        top_method
        | none
        | all_methods
        | all
        ]
      [ interface_selection
        none
        | simulation
        | all
        ]
      ]
    ]
  ]
  [ graphics ]
  [ check ]
  [ pre_run
    [ input STRING ]
    [ output STRING
      [ ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
        )
      | annotated
      | freeform ]
    ]
  ]

```

```

    ]
  ]
[ run
  [ input STRING ]
  [ output STRING ]
]
[ post_run
  [ input STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  [ output STRING ]
]
[ top_method_pointer ALIAS method_pointer STRING ]

KEYWORD method
[ id_method STRING ]
[ output
  debug
  | verbose
  | normal
  | quiet
  | silent
]
[ final_solutions INTEGER >= 0 ]
( hybrid
  ( sequential ALIAS uncoupled
    ( method_name_list STRINGLIST
      [ model_pointer_list STRINGLIST ]
    )
    | method_pointer_list STRINGLIST
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
  )
  |
  ( embedded ALIAS coupled
    ( global_method_name STRING
      [ global_model_pointer STRING ]
    )
    | global_method_pointer STRING
    ( local_method_name STRING
      [ local_model_pointer STRING ]
    )
    | local_method_pointer STRING
    [ local_search_probability REAL ]
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
  )
  |
  ( collaborative
    ( method_name_list STRINGLIST
      [ model_pointer_list STRINGLIST ]
    )
    | method_pointer_list STRINGLIST
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
  )
)

```

```

    )
  )
|
( multi_start
  ( method_name STRING
    [ model_pointer STRING ]
  )
  | method_pointer STRING
  [ random_starts INTEGER
    [ seed INTEGER ]
  ]
  [ starting_points REALLIST ]
  [ iterator_servers INTEGER > 0 ]
  [ iterator_scheduling
    master
    | peer
  ]
  [ processors_per_iterator INTEGER > 0 ]
)
|
( pareto_set
  ( method_name ALIAS opt_method_name STRING
    [ model_pointer ALIAS opt_model_pointer STRING ]
  )
  | method_pointer ALIAS opt_method_pointer STRING
  [ random_weight_sets INTEGER
    [ seed INTEGER ]
  ]
  [ weight_sets ALIAS multi_objective_weight_sets REALLIST ]
  [ iterator_servers INTEGER > 0 ]
  [ iterator_scheduling
    master
    | peer
  ]
  [ processors_per_iterator INTEGER > 0 ]
)
|
( branch_and_bound
  method_pointer STRING
  |
  ( method_name STRING
    [ model_pointer STRING ]
  )
  [ scaling ]
)
|
( surrogate_based_local
  method_pointer ALIAS approx_method_pointer STRING
  | method_name ALIAS approx_method_name STRING
  model_pointer ALIAS approx_model_pointer STRING
  [ soft_convergence_limit INTEGER ]
  [ truth_surrogate_bypass ]
  [ approx_subproblem
    original_primary
    | single_objective
    | augmented_lagrangian_objective
    | lagrangian_objective
  ]
  original_constraints
  | linearized_constraints
  | no_constraints
  ]
  [ merit_function
    penalty_merit
    | adaptive_penalty_merit
    | lagrangian_merit
    | augmented_lagrangian_merit
  ]
  [ acceptance_logic
    tr_ratio
    | filter
  ]
  [ constraint_relax
    homotopy

```

```

]
[ trust_region
  [ initial_size REALLIST ]
  [ minimum_size REAL ]
  [ contract_threshold REAL ]
  [ expand_threshold REAL ]
  [ contraction_factor REAL ]
  [ expansion_factor REAL ]
]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ constraint_tolerance REAL ]
)
|
( surrogate_based_global
method_pointer ALIAS approx_method_pointer STRING
| method_name ALIAS approx_method_name STRING
model_pointer ALIAS approx_model_pointer STRING
[ replace_points ]
[ max_iterations INTEGER >= 0 ]
)
|
( dot_frcg
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ constraint_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( dot_mmfd
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ constraint_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( dot_bfgs
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ constraint_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( dot_slp
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ constraint_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( dot_sqp
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ constraint_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( conmin_frcg

```



```

    [ max_iterations INTEGER >= 0 ]
    [ convergence_tolerance REAL ]
    [ constraint_tolerance REAL ]
    [ speculative ]
    [ max_function_evaluations INTEGER >= 0 ]
    [ scaling ]
    [ model_pointer STRING ]
  )
|
( conmin_mfd
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( dl_solver STRING
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( npsol_sqp
  [ verify_level INTEGER ]
  [ function_precision REAL ]
  [ linesearch_tolerance REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nlssol_sqp
  [ verify_level INTEGER ]
  [ function_precision REAL ]
  [ linesearch_tolerance REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nlpql_sqp
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_cg
  [ max_step REAL ]
  [ gradient_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_q_newton
  [ search_method
    value_based_line_search

```

```

    | gradient_based_line_search
    | trust_region
    | tr_pds
  ]
[ merit_function
  el_bakry
  | argaez_tapia
  | van_shanno
]
[ steplength_to_boundary REAL ]
[ centering_parameter REAL ]
[ max_step REAL ]
[ gradient_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( optpp_fd_newton
  [ search_method
    value_based_line_search
    | gradient_based_line_search
    | trust_region
    | tr_pds
  ]
  [ merit_function
    el_bakry
    | argaez_tapia
    | van_shanno
  ]
  [ steplength_to_boundary REAL ]
  [ centering_parameter REAL ]
  [ max_step REAL ]
  [ gradient_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_g_newton
  [ search_method
    value_based_line_search
    | gradient_based_line_search
    | trust_region
    | tr_pds
  ]
  [ merit_function
    el_bakry
    | argaez_tapia
    | van_shanno
  ]
  [ steplength_to_boundary REAL ]
  [ centering_parameter REAL ]
  [ max_step REAL ]
  [ gradient_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_newton
  [ search_method
    value_based_line_search
    | gradient_based_line_search

```

```

    | trust_region
    | tr_pds
  ]
[ merit_function
  el_bakry
  | argaez_tapia
  | van_shanno
]
[ steplength_to_boundary REAL ]
[ centering_parameter REAL ]
[ max_step REAL ]
[ gradient_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( optpp_pds
  [ search_scheme_size INTEGER ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( demo_tpl
  [ max_function_evaluations INTEGER >= 0 ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ options_file STRING ]
)
|
( rol
  [ max_iterations INTEGER >= 0 ]
  [ variable_tolerance REAL ]
  [ gradient_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ options_file STRING ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( asynch_pattern_search ALIAS coliny_apps
  [ initial_delta REAL ]
  [ contraction_factor REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ synchronization
    blocking
    | nonblocking
  ]
  [ merit_function
    merit_max
    | merit_max_smooth
    | merit1
    | merit1_smooth
    | merit2
    | merit2_smooth
    | merit2_squared
  ]
  [ constraint_penalty REAL ]
  [ smoothing_factor REAL ]
  [ constraint_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

```

```

|
( mesh_adaptive_search
  [ initial_delta REAL ]
  [ variable_tolerance REAL ]
  [ function_precision REAL ]
  [ seed INTEGER > 0 ]
  [ history_file STRING ]
  [ display_format STRING ]
  [ variable_neighborhood_search REAL ]
  [ neighbor_order INTEGER > 0 ]
  [ display_all_evaluations ]
  [ use_surrogate
    inform_search
    | optimize
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nowpac
  [ trust_region
    [ initial_size REALLIST ]
    [ minimum_size REAL ]
    [ contract_threshold REAL ]
    [ expand_threshold REAL ]
    [ contraction_factor REAL ]
    [ expansion_factor REAL ]
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( snowpac
  [ seed INTEGER > 0 ]
  [ trust_region
    [ initial_size REALLIST ]
    [ minimum_size REAL ]
    [ contract_threshold REAL ]
    [ expand_threshold REAL ]
    [ contraction_factor REAL ]
    [ expansion_factor REAL ]
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( moga
  [ fitness_type
    layer_rank
    | domination_count
  ]
  [ replacement_type
    elitist
    | roulette_wheel
    | unique_roulette_wheel
    |
    ( below_limit REAL
      [ shrinkage_fraction ALIAS shrinkage_percentage REAL ]
    )
  ]
  [ niching_type
    radial REALLIST
    | distance REALLIST
    |
    ( max_designs REALLIST
      [ num_designs INTEGER >= 2 ]
    )
  ]
)

```

```

]
[ convergence_type
  metric_tracker
  [ percent_change REAL ]
  [ num_generations INTEGER >= 0 ]
]
[ postprocessor_type
  orthogonal_distance REALLIST
]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ population_size INTEGER >= 0 ]
[ log_file STRING ]
[ print_each_pop ]
[ initialization_type
  simple_random
  | unique_random
  | flat_file STRING
]
[ crossover_type
  multi_point_binary INTEGER
  | multi_point_parameterized_binary INTEGER
  | multi_point_real INTEGER
  |
  ( shuffle_random
    [ num_parents INTEGER > 0 ]
    [ num_offspring INTEGER > 0 ]
  )
  [ crossover_rate REAL ]
]
[ mutation_type
  bit_random
  | replace_uniform
  |
  ( offset_normal
    [ mutation_scale REAL ]
  )
  |
  ( offset_cauchy
    [ mutation_scale REAL ]
  )
  |
  ( offset_uniform
    [ mutation_scale REAL ]
  )
  [ mutation_rate REAL ]
]
[ seed INTEGER > 0 ]
[ convergence_tolerance REAL ]
[ model_pointer STRING ]
)
|
( soga
  [ fitness_type
    merit_function
    [ constraint_penalty REAL ]
  ]
  [ replacement_type
    elitist
    | favor_feasible
    | roulette_wheel
    | unique_roulette_wheel
  ]
  [ convergence_type
    ( best_fitness_tracker
      [ percent_change REAL ]
      [ num_generations INTEGER >= 0 ]
    )
    |
    ( average_fitness_tracker
      [ percent_change REAL ]
      [ num_generations INTEGER >= 0 ]
    )
  ]
)

```

```

    )
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ population_size INTEGER >= 0 ]
  [ log_file STRING ]
  [ print_each_pop ]
  [ initialization_type
    simple_random
    | unique_random
    | flat_file STRING
  ]
  [ crossover_type
    multi_point_binary INTEGER
    | multi_point_parameterized_binary INTEGER
    | multi_point_real INTEGER
    |
    ( shuffle_random
      [ num_parents INTEGER > 0 ]
      [ num_offspring INTEGER > 0 ]
    )
    [ crossover_rate REAL ]
  ]
  [ mutation_type
    bit_random
    | replace_uniform
    |
    ( offset_normal
      [ mutation_scale REAL ]
    )
    |
    ( offset_cauchy
      [ mutation_scale REAL ]
    )
    |
    ( offset_uniform
      [ mutation_scale REAL ]
    )
    [ mutation_rate REAL ]
  ]
  [ seed INTEGER > 0 ]
  [ convergence_tolerance REAL ]
  [ model_pointer STRING ]
)
|
( colony_pattern_search
  [ constant_penalty ]
  [ no_expansion ]
  [ expand_after_success INTEGER ]
  [ pattern_basis
    coordinate
    | simplex
  ]
  [ stochastic ]
  [ total_pattern_size INTEGER ]
  [ exploratory_moves
    multi_step
    | adaptive_pattern
    | basic_pattern
  ]
  [ synchronization
    blocking
    | nonblocking
  ]
  [ contraction_factor REAL ]
  [ constraint_penalty REAL ]
  [ initial_delta REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
)

```

```

[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( coliny_solis_wets
[ contract_after_failure INTEGER ]
[ no_expansion ]
[ expand_after_success INTEGER ]
[ constant_penalty ]
[ contraction_factor REAL ]
[ constraint_penalty REAL ]
[ initial_delta REAL ]
[ variable_tolerance REAL ]
[ solution_target ALIAS solution_accuracy REAL ]
[ seed INTEGER > 0 ]
[ show_misc_options ]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( coliny_cobyla
[ initial_delta REAL ]
[ variable_tolerance REAL ]
[ solution_target ALIAS solution_accuracy REAL ]
[ seed INTEGER > 0 ]
[ show_misc_options ]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( coliny_direct
[ division
  major_dimension
  | all_dimensions
]
[ global_balance_parameter REAL ]
[ local_balance_parameter REAL ]
[ max_boxsize_limit REAL ]
[ min_boxsize_limit REAL ]
[ constraint_penalty REAL ]
[ solution_target ALIAS solution_accuracy REAL ]
[ seed INTEGER > 0 ]
[ show_misc_options ]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( coliny_ea
[ population_size INTEGER > 0 ]
[ initialization_type
  simple_random
  | unique_random
  | flat_file STRING
]
[ fitness_type
  linear_rank
  | merit_function
]
)

```

```

[ replacement_type
  random INTEGER
  | chc INTEGER
  | elitist INTEGER
  [ new_solutions_generated INTEGER ]
]
[ crossover_rate REAL ]
[ crossover_type
  two_point
  | blend
  | uniform
]
[ mutation_rate REAL ]
[ mutation_type
  replace_uniform
  |
  ( offset_normal
    [ mutation_scale REAL ]
    [ mutation_range INTEGER ]
  )
  |
  ( offset_cauchy
    [ mutation_scale REAL ]
    [ mutation_range INTEGER ]
  )
  |
  ( offset_uniform
    [ mutation_scale REAL ]
    [ mutation_range INTEGER ]
  )
  [ non_adaptive ]
]
[ constraint_penalty REAL ]
[ solution_target ALIAS solution_accuracy REAL ]
[ seed INTEGER > 0 ]
[ show_misc_options ]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( coliny_beta
  beta_solver_name STRING
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nl2sol
  [ function_precision REAL ]
  [ absolute_conv_tol REAL ]
  [ x_conv_tol REAL ]
  [ singular_conv_tol REAL ]
  [ singular_radius REAL ]
  [ false_conv_tol REAL ]
  [ initial_trust_radius REAL ]
  [ covariance INTEGER ]
  [ regression_diagnostics ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

```



```

)
|
( nonlinear_cg
  [ misc_options STRINGLIST ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( ncsu_direct
  [ solution_target ALIAS solution_accuracy REAL ]
  [ min_boxsize_limit REAL ]
  [ volume_boxsize_limit REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( genie_opt_darts
  [ seed INTEGER > 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( genie_direct
  [ seed INTEGER > 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( efficient_global
  [ initial_samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ batch_size INTEGER >= 1
    [ exploration INTEGER >= 0 ]
    [ synchronization
      blocking
      | nonblocking
    ]
  ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ x_conv_tol REAL ]
  [ gaussian_process ALIAS kriging
    ( surfpack
      [ export_model ]
      [ filename_prefix STRING ]
      ( formats
        [ text_archive ]
        [ binary_archive ]
      )
    )
  ]
  )
| dakota
|
( experimental
  [ export_model ]
  [ filename_prefix STRING ]
  ( formats
    [ text_archive ]
    [ binary_archive ]
  )
)
| [ options_file STRING ]
)
|
[ use_derivatives ]
[ import_build_points_file ALIAS import_points_file STRING ]

```

```

    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ model_pointer STRING ]
)
|
( surrogate_based_uq
  [ samples_on_emulator ALIAS samples INTEGER ]
  [ sample_type
    lhs
    | random
  ]
  [ rng
    mt19937
    | rnum2
  ]
  [ probability_refinement ALIAS sample_refinement
    import
    | adapt_import
    | mm_adapt_import
    [ refinement_samples INTEGERLIST ]
  ]
  [ final_moments
    none
    | standard
    | central
  ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | reliabilities
      | gen_reliabilities
      [ system
        series
        | parallel
      ]
    ]
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ]
  ]
  [ reliability_levels REALLIST
    [ num_reliability_levels INTEGERLIST ]
  ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ variance_based_decomp
    [ interaction_order INTEGER > 0 ]
    [ drop_tolerance REAL ]
  ]
  [ diagonal_covariance
  | full_covariance ]
)

```

```

[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
[ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ seed INTEGER > 0 ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( function_train
  [ p_refinement
    ( uniform
      increment_start_rank
      | increment_start_order
      | increment_max_rank
      | increment_max_order
      | increment_max_rank_order
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ regression_type
    ls
    |
    ( rls2
      l2_penalty REAL
    )
  ]
  [ max_solver_iterations INTEGER >= 0 ]
  [ max_cross_iterations INTEGER >= 0 ]
  [ solver_tolerance REAL ]
  [ response_scaling ]
  [ tensor_grid ]
  collocation_points INTEGER
  | collocation_ratio REAL
  [ rounding_tolerance REAL ]
  [ arithmetic_tolerance REAL ]
  [ start_order ALIAS order INTEGER >= 0
    [ dimension_preference REALLIST ]
  ]
  [ adapt_order ]
  [ kick_order INTEGER > 0 ]
  [ max_order INTEGER >= 0 ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  [ start_rank ALIAS rank INTEGER >= 0 ]
  [ adapt_rank ]
  [ kick_rank INTEGER > 0 ]
  [ max_rank INTEGER >= 0 ]
  [ max_cv_rank_candidates INTEGER >= 0 ]
  [ samples_on_emulator ALIAS samples INTEGER ]
  [ sample_type
    lhs
    | random
  ]
]

```

```

[ rng
  mt19937
  | rnum2
]
[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ seed INTEGER > 0 ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( multifidelity_function_train
  [ p_refinement
    ( uniform

```

```

    increment_start_rank
    | increment_start_order
    | increment_max_rank
    | increment_max_order
    | increment_max_rank_order
  )
]
[ max_refinement_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ metric_scale
  relative
  | absolute
]
[ statistics_mode
  active
  | combined
]
[ allocation_control
  greedy
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
[ rounding_tolerance REAL ]
[ arithmetic_tolerance REAL ]
[ regression_type
  ls
  |
  ( rls2
    l2_penalty REAL
  )
]
[ max_solver_iterations INTEGER >= 0 ]
[ max_cross_iterations INTEGER >= 0 ]
[ solver_tolerance REAL ]
[ response_scaling ]
[ tensor_grid ]
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ collocation_ratio REAL ]
[ start_order_sequence ALIAS order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
]
[ adapt_order ]
[ kick_order INTEGER > 0 ]
[ max_order INTEGER >= 0 ]
[ max_cv_order_candidates INTEGER >= 0 ]
[ start_rank_sequence ALIAS rank_sequence INTEGERLIST ]
[ adapt_rank ]
[ kick_rank INTEGER > 0 ]
[ max_rank INTEGER >= 0 ]
[ max_cv_rank_candidates INTEGER >= 0 ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
]
[ rng
  mt19937
  | rnum2
]
[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST

```

```

    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | reliabilities
      | gen_reliabilities
      [ system
series
| parallel
]
]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( multilevel_function_train
  [ max_iterations INTEGER >= 0 ]
  [ allocation_control
    ( estimator_variance
      [ estimator_rate REAL ]
    )
  | rank_sampling
]
[ convergence_tolerance REAL ]
[ metric_scale
  relative
  | absolute
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
[ rounding_tolerance REAL ]
[ arithmetic_tolerance REAL ]

```

```

[ regression_type
  ls
  |
  ( rls2
    l2_penalty REAL
  )
]
[ max_solver_iterations INTEGER >= 0 ]
[ max_cross_iterations INTEGER >= 0 ]
[ solver_tolerance REAL ]
[ response_scaling ]
[ tensor_grid ]
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ collocation_ratio REAL ]
[ start_order_sequence ALIAS order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
]
[ adapt_order ]
[ kick_order INTEGER > 0 ]
[ max_order INTEGER >= 0 ]
[ max_cv_order_candidates INTEGER >= 0 ]
[ start_rank_sequence ALIAS rank_sequence INTEGERLIST ]
[ adapt_rank ]
[ kick_rank INTEGER > 0 ]
[ max_rank INTEGER >= 0 ]
[ max_cv_rank_candidates INTEGER >= 0 ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
]
[ rng
  mt19937
  | rnum2
]
[ probability_refinement ALIAS sample_refinement
import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
]

```

```

    [ drop_tolerance REAL ]
  ]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( polynomial_chaos ALIAS nond_polynomial_chaos
  [ p_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | decay
      | generalized
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  ( quadrature_order INTEGER
    [ dimension_preference REALLIST ]
    [ nested
    | non_nested ]
  )
  |
  ( sparse_grid_level INTEGER
    [ dimension_preference REALLIST ]
    [ restricted
    | unrestricted ]
    [ nested
    | non_nested ]
  )
  | cubature_integrand INTEGER
  |
  ( expansion_order INTEGER
    [ dimension_preference REALLIST ]
    [ basis_type
      tensor_product
      | total_order
    ]
    |
    ( adapted
      [ advancements INTEGER ]
      [ soft_convergence_limit INTEGER ]
    )
  ]
  ( collocation_points INTEGER
    [ ( least_squares
      [ svd

```



```

    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
  |
  ( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
  |
  ( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
  [ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( collocation_ratio REAL
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
  |
  ( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
  |
  ( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
  [ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples INTEGER
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]

```

```

    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
[ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
collocation_points INTEGER
[ tensor_grid INTEGERLIST ]
[ reuse_points ALIAS reuse_samples ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
| import_expansion_file STRING
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
]
[ rng
  mt19937
  | rnum2
]
[ probability_refinement ALIAS sample_refinement
import
| adapt_import
| mm_adapt_import
[ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
[ num_response_levels INTEGERLIST ]
[ compute
  probabilities
  | reliabilities
  | gen_reliabilities
  [ system
series
| parallel
]
]
]
[ probability_levels REALLIST
[ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
[ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
[ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary

```

```

]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ seed INTEGER > 0 ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( multifidelity_polynomial_chaos
  [ p_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | decay
      | generalized
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ statistics_mode
    active
    | combined
  ]
  [ allocation_control
    greedy
  ]
  [ discrepancy_emulation
    distinct ALIAS paired
    | recursive
  ]
  ( quadrature_order_sequence INTEGERLIST
    [ dimension_preference REALLIST ]
    [ nested
    | non_nested ]
  )
  |
  ( sparse_grid_level_sequence INTEGERLIST
    [ dimension_preference REALLIST ]
    [ restricted
    | unrestricted ]
    [ nested
    | non_nested ]
  )
  |
  ( expansion_order_sequence INTEGERLIST

```

```

[ dimension_preference REALLIST ]
[ basis_type
  tensor_product
  | total_order
  |
  ( adapted
[ advancements INTEGER ]
[ soft_convergence_limit INTEGER ]
)
]
( collocation_ratio REAL
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ ( least_squares
    [ svd
    | equality_constrained ]
    )
  |
  ( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
  |
  ( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
  |
  ( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
  [ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
  )
  |
  ( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
  )
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
collocation_points_sequence INTEGERLIST
[ tensor_grid INTEGERLIST ]
[ reuse_points ALIAS reuse_samples ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]

```

```

    [ active_only ]
  ]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
]
[ rng
  mt19937
  | rnum2
]
[ probability_refinement ALIAS sample_refinement
import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  ]
]

```

```

    )
    | annotated
    | freeform ]
  ]
  [ seed_sequence INTEGERLIST ]
  [ fixed_seed ]
  [ model_pointer STRING ]
)
|
( multilevel_polynomial_chaos
  [ max_iterations INTEGER >= 0 ]
  [ allocation_control
    ( estimator_variance
      [ estimator_rate REAL ]
    )
    | rip_sampling
  ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ discrepancy_emulation
    distinct ALIAS paired
    | recursive
  ]
  ( expansion_order_sequence INTEGERLIST
    [ dimension_preference REALLIST ]
    [ basis_type
      tensor_product
      | total_order
      |
      ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
  ]
  ( collocation_ratio REAL
    [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
    [ ( least_squares
      [ svd
      | equality_constrained ]
    )
    |
    ( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
  ]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
  ]
  |
  ( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
  ]
  |
  ( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
  ]
  [ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
  )
)
|

```

```

( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
  ]
[ rng
  mt19937
  | rnum2
  ]
[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
  ]
[ final_moments
  none
  | standard
  | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
  ]
  series
  | parallel
  ]
  ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST

```

```

    [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( stoch_collocation ALIAS nond_stoch_collocation
  [ ( p_refinement
    uniform
    |
    ( dimension_adaptive
    sobol
    | generalized
    )
  )
  |
  ( h_refinement
    uniform
    |
    ( dimension_adaptive
    sobol
    | generalized
    )
    | local_adaptive
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  ( quadrature_order INTEGER
    [ dimension_preference REALLIST ]
    [ nested
    | non_nested ]
  )
  |
  ( sparse_grid_level INTEGER
    [ dimension_preference REALLIST ]
    [ nodal
    | hierarchical ]
    [ restricted
    | unrestricted ]
  )
)

```



```

    [ nested
    | non_nested ]
  )
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
]
[ rng
  mt19937
  | rnum2
]
[ probability_refinement ALIAS sample_refinement
import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
  series
  | parallel
  ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]

```

```

    )
    | annotated
    | freeform ]
  ]
  [ seed INTEGER > 0 ]
  [ fixed_seed ]
  [ model_pointer STRING ]
)
|
( multifidelity_stoch_collocation
  [ ( p_refinement
    uniform
    |
    ( dimension_adaptive
    sobol
    | generalized
    )
    )
  |
  ( h_refinement
  uniform
  |
  ( dimension_adaptive
  sobol
  | generalized
  )
  | local_adaptive
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
  relative
  | absolute
  ]
  [ statistics_mode
  active
  | combined
  ]
  [ allocation_control
  greedy
  ]
  [ discrepancy_emulation
  distinct ALIAS paired
  | recursive
  ]
  ( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
  )
  |
  ( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
  )
  [ piecewise
  | askey
  | wiener ]
  [ use_derivatives ]
  [ samples_on_emulator ALIAS samples INTEGER ]
  [ sample_type
  lhs
  | random
  ]
  [ rng
  mt19937
  | rnum2
  ]
]

```

```

[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( sampling ALIAS nond_sampling
  [ samples ALIAS initial_samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ fixed_seed ]
  [ sample_type
    lhs
    | random
  ]
)

```

```

    | incremental_lhs
    | incremental_random
  ]
[ refinement_samples INTEGERLIST ]
[ d_optimal
  [ candidate_designs INTEGER > 0
  | leja_oversample_ratio REAL ]
]
[ variance_based_decomp
  [ drop_tolerance REAL ]
]
[ backfill ]
[ principal_components
  [ percent_variance_explained REAL ]
]
[ wilks
  [ order INTEGER ]
  [ confidence_level REAL ]
  [ one_sided_lower ]
  [ one_sided_upper ]
  [ two_sided ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( multilevel_sampling ALIAS multilevel_mc ALIAS mlmc
  [ seed_sequence INTEGERLIST ]
  [ fixed_seed ]
  [ pilot_samples ALIAS initial_samples INTEGERLIST ]
  [ solution_mode
    online_pilot
    | offline_pilot
    | pilot_projection
  ]
  [ sample_type
    lhs
    | random
  ]
  [ export_sample_sequence

```

```

    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  [ allocation_target
    mean
    |
    ( variance
      [ optimization ]
    )
    |
    ( standard_deviation
      [ optimization ]
    )
    |
    ( scalarization
      [ scalarization_response_mapping REALLIST ]
      [ optimization ]
    )
  ]
  [ qoi_aggregation
    sum
    | max
  ]
  [ convergence_tolerance REAL ]
  [ convergence_tolerance_type
    relative
    | absolute
  ]
  [ convergence_tolerance_target
    variance_constraint
    | cost_constraint
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ final_statistics
    estimator_performance
    |
    ( qoi_statistics
      [ final_moments
        none
        | standard
        | central
      ]
      [ distribution
        cumulative
        | complementary
      ]
    )
  ]
  [ rng
    mtl19937
    | rnum2
  ]
  [ model_pointer STRING ]
)
|
( multifidelity_sampling ALIAS multifidelity_mc ALIAS mfmc
  [ seed_sequence INTEGERLIST ]
  [ fixed_seed ]
  [ pilot_samples ALIAS initial_samples INTEGERLIST ]
  [ solution_mode
    online_pilot
    | offline_pilot
    | pilot_projection
  ]
  [ numerical_solve
    [ fallback
    | override ]
  ]
)

```

```

    [ sqp
    | nip ]
  ]
[ sample_type
  lhs
  | random
  ]
[ export_sample_sequence
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  ]
[ convergence_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ final_statistics
  estimator_performance
  |
  ( qoi_statistics
    [ final_moments
  none
  | standard
  | central
  ]
  [ distribution
  cumulative
  | complementary
  ]
  )
  ]
[ rng
  mt19937
  | rnum2
  ]
[ model_pointer STRING ]
)
|
( multilevel_multifidelity_sampling ALIAS multilevel_multifidelity_mc ALIAS mlmfmc
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ pilot_samples ALIAS initial_samples INTEGERLIST ]
[ solution_mode
  online_pilot
  | offline_pilot
  | pilot_projection
  ]
[ sample_type
  lhs
  | random
  ]
[ export_sample_sequence
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  ]
[ convergence_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ final_statistics
  estimator_performance
  |
  ( qoi_statistics
    [ final_moments
  none
  | standard

```

```

    | central
    ]
    [ distribution
    cumulative
    | complementary
    ]
    )
  ]
[ rng
  mt19937
  | rnum2
  ]
[ model_pointer STRING ]
)
|
( approximate_control_variate ALIAS acv_sampling
  acv_independent_sampling ALIAS acv_is
  | acv_multifidelity ALIAS acv_mf
  | acv_k1 ALIAS acv_adaptive
  [ pilot_samples ALIAS initial_samples INTEGERLIST ]
  [ solution_mode
    online_pilot
    | offline_pilot
    | pilot_projection
  ]
  [ truth_fixed_by_pilot ]
  [ sqp
  | nip ]
  [ seed_sequence INTEGERLIST ]
  [ fixed_seed ]
  [ sample_type
    lhs
    | random
  ]
  [ export_sample_sequence
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ final_statistics
    estimator_performance
    |
    ( qoi_statistics
      [ final_moments
    none
    | standard
    | central
    ]
    [ distribution
    cumulative
    | complementary
    ]
    )
  ]
  [ rng
    mt19937
    | rnum2
  ]
  [ model_pointer STRING ]
  )
|
( importance_sampling ALIAS nond_importance_sampling
  [ samples ALIAS initial_samples INTEGER ]
  [ seed INTEGER > 0 ]
  import
  | adapt_import

```

```

| mm_adapt_import
[ refinement_samples INTEGERLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
      series
      | parallel
      ]
    ]
  ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ distribution
  cumulative
  | complementary
  ]
[ rng
  mt19937
  | rnum2
  ]
[ model_pointer STRING ]
)
|
( gpais ALIAS gaussian_process_adaptive_importance_sampling
[ build_samples ALIAS samples INTEGER ]
[ seed INTEGER > 0 ]
[ samples_on_emulator INTEGER ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  ]
[ max_iterations INTEGER >= 0 ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
      series
      | parallel
      ]
    ]
  ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
)

```



```

[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( adaptive_sampling ALIAS nond_adaptive_sampling
[ initial_samples ALIAS samples INTEGER ]
[ seed INTEGER > 0 ]
[ samples_on_emulator INTEGER ]
[ fitness_metric
  predicted_variance
  | distance
  | gradient
]
[ batch_selection
  naive
  | distance_penalty
  | topology
  | constant_liar
]
[ refinement_samples INTEGERLIST ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
      series
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]

```

```

    [ model_pointer STRING ]
  )
|
( pof_darts ALIAS nond_pof_darts
  build_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ lipschitz
    local
    | global
  ]
  [ samples_on_emulator INTEGER ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ]
  ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ rng
    mt19937
    | rnum2
  ]
  [ model_pointer STRING ]
)
|
( rkd_darts ALIAS nond_rkd_darts
  build_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ lipschitz
    local
    | global
  ]
  [ samples_on_emulator INTEGER ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ]
  ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ rng
    mt19937
    | rnum2
  ]
  [ model_pointer STRING ]
)

```

```

)
|
( global_evidence ALIAS nond_global_evidence
  [ samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ ( sbgo
    [ gaussian_process ALIAS kriging
    surfpack
    | dakota
    |
    ( experimental
      [ options_file STRING ]
    )
  )
  [ use_derivatives ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
  [ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  ]
)
|
( ego
  [ gaussian_process ALIAS kriging
  ( surfpack
  [ export_model
    [ filename_prefix STRING ]
    ( formats
      [ text_archive ]
      [ binary_archive ]
    )
  )
  ]
)
| dakota
|
( experimental
[ export_model
  [ filename_prefix STRING ]
  ( formats
    [ text_archive ]
    [ binary_archive ]
  )
]
[ options_file STRING ]
)
]
[ use_derivatives ]
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
)
| annotated
| freeform ]
[ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
[ ( custom_annotated

```

```

    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
| ea
| lhs ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( global_interval_est ALIAS nond_global_interval_est
  [ samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ ( sbgo
    [ gaussian_process ALIAS kriging
      surfpack
      | dakota
      |
      ( experimental
        [ options_file STRING ]
      )
    ]
    [ use_derivatives ]
    [ import_build_points_file ALIAS import_points_file STRING
  ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
    | annotated
    | freeform ]
  [ active_only ]
  ]
  [ export_approx_points_file ALIAS export_points_file STRING
  ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
]

```

```

    )
  |
  ( ego
    [ gaussian_process ALIAS kriging
      ( surfpack
        [ export_model
          [ filename_prefix STRING ]
          ( formats
            [ text_archive ]
            [ binary_archive ]
          )
        ]
      )
    | dakota
    |
    ( experimental
      [ export_model
        [ filename_prefix STRING ]
        ( formats
          [ text_archive ]
          [ binary_archive ]
        )
      ]
    [ options_file STRING ]
  )
  ]
  [ use_derivatives ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ export_approx_points_file ALIAS export_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  ]
  | ea
  | lhs ]
  [ rng
    mt19937
    | rnum2
  ]
  [ model_pointer STRING ]
)
|
( bayes_calibration ALIAS nond_bayes_calibration
  ( queso
    chain_samples ALIAS samples INTEGER
    [ seed INTEGER > 0 ]
    [ rng
      mt19937
      | rnum2
    ]
  [ emulator
    ( gaussian_process ALIAS kriging
      surfpack
      | dakota
      [ build_samples INTEGER ]
      [ posterior_adaptive ]
      [ import_build_points_file ALIAS import_points_file STRING
        [ ( custom_annotated
          [ header ]

```

```

        [ eval_id ]
        [ interface_id ]
    )
| annotated
| freeform ]
[ active_only ]
]
)
|
( pce
[ p_refinement
uniform
|
( dimension_adaptive
sobol
| decay
| generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
( quadrature_order INTEGER
[ dimension_preference REALLIST ]
[ nested
| non_nested ]
)
|
( sparse_grid_level INTEGER
[ dimension_preference REALLIST ]
[ restricted
| unrestricted ]
[ nested
| non_nested ]
)
| cubature_integrand INTEGER
|
( expansion_order INTEGER
[ dimension_preference REALLIST ]
[ basis_type
tensor_product
| total_order
|
( adapted
[ advancements INTEGER ]
[ soft_convergence_limit INTEGER ]
)
]
)
( collocation_points INTEGER
[ ( least_squares
[ svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
[ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
]

```

```

    [ ratio_order REAL ]
    [ response_scaling ]
    [ use_derivatives ]
    [ tensor_grid ]
    [ reuse_points ALIAS reuse_samples ]
    [ max_solver_iterations INTEGER >= 0 ]
  )
|
( collocation_ratio REAL
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  )
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  )
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples INTEGER
  [ reuse_points ALIAS reuse_samples ]
)
|
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
|
[ posterior_adaptive ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points INTEGER
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)

```

```

    [ posterior_adaptive ]
  )
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( ml_pce
[ allocation_control
( estimator_variance
  [ estimator_rate REAL ]
)
| rip_sampling
]
[ discrepancy_emulation
distinct ALIAS paired
| recursive
]
expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
  tensor_product
  | total_order
  |
  ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
]
( collocation_ratio REAL
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ ( least_squares
[ svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
]
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
[ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated

```



```

    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
    ]
  )
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
collocation_points_sequence INTEGERLIST
[ tensor_grid INTEGERLIST ]
[ reuse_points ALIAS reuse_samples ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_pce
[ p_refinement
uniform
|
( dimension_adaptive
sobol
| decay
| generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
greedy
]
[ discrepancy_emulation
distinct ALIAS paired
| recursive
]
( quadrature_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ nested
| non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ restricted
| unrestricted ]
[ nested
| non_nested ]
)
|
( expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
tensor_product
| total_order
|
( adapted
[ advancements INTEGER ]

```

```

    [ soft_convergence_limit INTEGER ]
  )
)
( collocation_ratio REAL
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  )
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  )
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
)
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]

```

```

[ diagonal_covariance
| full_covariance ]
)
|
( sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
| local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
| local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
  greedy
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive

```

```

]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
]
[ standardized_space ]
[ logit_transform ]
[ export_chain_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ dram
| delayed_rejection
| adaptive_metropolis
| metropolis_hastings
| multilevel ]
[ pre_solve
  sqp
  | nip
  | none
]
[ proposal_covariance
  ( prior
[ multiplier REAL > 0.0 ]
)
|
  ( derivatives
[ update_period INTEGER ]
)
|
  ( values REALLIST
diagonal
| matrix
)
|
  ( filename STRING
diagonal
| matrix
)
]
[ options_file STRING ]
)
|
( gpmsa
chain_samples ALIAS samples INTEGER
[ seed INTEGER > 0 ]
[ rng
  mt19937
  | rnum2

```

```

]
build_samples INTEGER
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ standardized_space ]
[ logit_transform ]
[ gpmsa_normalize ]
[ export_chain_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ dram
| delayed_rejection
| adaptive_metropolis
| metropolis_hastings ]
[ proposal_covariance
  ( prior
[ multiplier REAL > 0.0 ]
)
|
  ( derivatives
[ update_period INTEGER ]
)
|
  ( values REALLIST
diagonal
| matrix
)
|
  ( filename STRING
diagonal
| matrix
)
]
[ options_file STRING ]
)
|
( wasabi
pushforward_samples INTEGER
[ seed INTEGER > 0 ]
[ emulator
  ( gaussian_process ALIAS kriging
surfpack
| dakota
[ build_samples INTEGER ]
[ posterior_adaptive ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
[ active_only ]
]
)
|
  ( pce
[ p_refinement
uniform

```

```

|
( dimension_adaptive
  sobol
  | decay
  | generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
|
cubature_integrand INTEGER
|
( expansion_order INTEGER
  [ dimension_preference REALLIST ]
  [ basis_type
  tensor_product
  | total_order
  |
  ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
  ]
)
|
( collocation_points INTEGER
  [ ( least_squares
  | svd
  | equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
)
|
basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
|
( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
)
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
)
|
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
|
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( collocation_ratio REAL
  [ ( least_squares
  | svd
  | equality_constrained ]
)

```

```

)
|
( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
)
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
|
( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
)
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
)
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples INTEGER
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ posterior_adaptive ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points INTEGER
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
  [ posterior_adaptive ]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( ml_pce
[ allocation_control

```

```

( estimator_variance
  [ estimator_rate REAL ]
)
| rip_sampling
]
[ discrepancy_emulation
distinct ALIAS paired
| recursive
]
( expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
  tensor_product
  | total_order
  |
  ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
]
)
( collocation_ratio REAL
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ ( least_squares
[ svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
[ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
[ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
[ header ]
[ eval_id ]
[ interface_id ]
)
| annotated
| freeform ]
[ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
collocation_points_sequence INTEGERLIST

```



```

[ tensor_grid INTEGERLIST ]
[ reuse_points ALIAS reuse_samples ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_pce
[ p_refinement
uniform
|
( dimension_adaptive
sobol
| decay
| generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
greedy
]
[ discrepancy_emulation
distinct ALIAS paired
| recursive
]
( quadrature_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ nested
| non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ restricted
| unrestricted ]
[ nested
| non_nested ]
)
|
( expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
tensor_product
| total_order
|
( adapted
[ advancements INTEGER ]
[ soft_convergence_limit INTEGER ]
)
]
)
[ collocation_ratio REAL
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ ( least_squares
[ svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]

```

```

]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
|
( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
)
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
)
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)

```

```

    )
  |
  ( h_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | generalized
    )
    | local_adaptive
  ]
[ max_refinement_iterations INTEGER >= 0 ]
( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
greedy
]
[ discrepancy_emulation
distinct ALIAS paired
| recursive
]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted

```

```

    | unrestricted ]
    [ nested
    | non_nested ]
    )
  [ piecewise
  | askey
  | wiener ]
  [ use_derivatives ]
  [ diagonal_covariance
  | full_covariance ]
  )
  ]
  [ standardized_space ]
  ( data_distribution
  ( gaussian
means REALLIST
  ( covariance REALLIST
  diagonal
  | matrix
  )
  )
  | obs_data_filename STRING
  )
  [ posterior_samples_import_filename STRING ]
  [ generate_posterior_samples
  [ posterior_samples_export_filename STRING ]
  ]
  [ evaluate_posterior_density
  [ posterior_density_export_filename STRING ]
  ]
  )
  |
  ( dream
  chain_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ chains INTEGER >= 3 ]
  [ num_cr INTEGER >= 1 ]
  [ crossover_chain_pairs INTEGER >= 0 ]
  [ gr_threshold REAL > 0.0 ]
  [ jump_step INTEGER >= 0 ]
  [ emulator
  ( gaussian_process ALIAS kriging
surfpack
  | dakota
  [ build_samples INTEGER ]
  [ posterior_adaptive ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
  )
  |
  ( pce
  [ p_refinement
  uniform
  |
  ( dimension_adaptive
  sobol
  | decay
  | generalized
  )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  ( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]

```

```

)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
| cubature_integrand INTEGER
|
( expansion_order INTEGER
  [ dimension_preference REALLIST ]
  [ basis_type
    tensor_product
    | total_order
  ]
  ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
)
|
( collocation_points INTEGER
  [ ( least_squares
  | svd
  | equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
)
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
|
( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
)
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
)
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( collocation_ratio REAL
  [ ( least_squares
  | svd
  | equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
)
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
|
( least_angle_regression ALIAS lars

```

```

    [ noise_tolerance REALLIST ]
  ]
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
  ]
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  ]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples INTEGER
  [ reuse_points ALIAS reuse_samples ]
  )
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ posterior_adaptive ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points INTEGER
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
    | annotated
    | freeform ]
    [ active_only ]
    ]
  [ posterior_adaptive ]
  )
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( ml_pce
[ allocation_control
  ( estimator_variance
    [ estimator_rate REAL ]
    )
  | rip_sampling
  ]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
  ]
( expansion_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ basis_type

```

```

    tensor_product
    | total_order
    |
    ( adapted
      [ advancements INTEGER ]
      [ soft_convergence_limit INTEGER ]
    )
  ]
( collocation_ratio REAL
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ ( least_squares
    [ svd
    | equality_constrained ]
    )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  )
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  )
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)

```

```

)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_pce
[ p_refinement
uniform
|
( dimension_adaptive
sobol
| decay
| generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
greedy
]
[ discrepancy_emulation
distinct ALIAS paired
| recursive
]
( quadrature_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ nested
| non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ restricted
| unrestricted ]
[ nested
| non_nested ]
)
|
( expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
tensor_product
| total_order
|
( adapted
[ advancements INTEGER ]
[ soft_convergence_limit INTEGER ]
)
]
( collocation_ratio REAL
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ ( least_squares
| svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso

```



```

    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  ]
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
)
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]

```

```

( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
  greedy
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
]

```

```

[ standardized_space ]
[ export_chain_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
)
|
( muq
  chain_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ rng
    mt19937
    | rnum2
    ]
  [ dram
  | delayed_rejection
  | adaptive_metropolis
  | metropolis_hastings ]
  [ proposal_covariance
    ( prior
  [ multiplier REAL > 0.0 ]
    )
  |
  ( derivatives
  [ update_period INTEGER ]
    )
  |
  ( values REALLIST
  diagonal
  | matrix
  )
  |
  ( filename STRING
  diagonal
  | matrix
  )
  ]
)
[ experimental_design
  initial_samples ALIAS samples INTEGER
  num_candidates INTEGER > 0
  [ max_hifi_evaluations INTEGER >= 0 ]
  [ batch_size INTEGER >= 1 ]
  [ import_candidate_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
    | annotated
    | freeform ]
  ]
  [ ksg2 ]
]
[ calibrate_error_multipliers
  one
  | per_experiment
  | per_response
  | both
  [ hyperprior_alphas REALLIST
  hyperprior_betas REALLIST
  ]
]
[ burn_in_samples INTEGER ]
[ posterior_stats
  [ kl_divergence ]
  [ mutual_info
  [ ksg2 ]

```

```

    ]
  [ kde ]
]
[ chain_diagnostics
  [ confidence_intervals ]
]
[ model_evidence
  [ mc_approx ]
  [ evidence_samples INTEGER ]
  [ laplace_approx ]
]
[ model_discrepancy
  [ discrepancy_type
    ( gaussian_process ALIAS kriging
  [ trend_order INTEGER >= 0 ]
    )
    |
    ( polynomial
  [ basis_order INTEGER >= 0 ]
    )
  ]
  [ num_prediction_configs INTEGER >= 0 ]
  [ prediction_configs REALLIST ]
  [ import_prediction_configs STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ export_discrepancy_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ export_corrected_model_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ export_corrected_variance_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
]
[ sub_sampling_period INTEGER ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ convergence_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ model_pointer STRING ]
[ scaling ]
)
|
( dace
  grid
  | random

```

```

| oas
| lhs
| oa_lhs
| box_behnken
| central_composite
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ fixed_seed ]
[ main_effects ]
[ quality_metrics ]
[ variance_based_decomp
  [ drop_tolerance REAL ]
]
[ symbols INTEGER ]
[ model_pointer STRING ]
)
|
( fsu_cvt
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ fixed_seed ]
[ latinize ]
[ quality_metrics ]
[ variance_based_decomp
  [ drop_tolerance REAL ]
]
[ trial_type
  grid
  | halton
  | random
]
[ num_trials INTEGER ]
[ max_iterations INTEGER >= 0 ]
[ model_pointer STRING ]
)
|
( psuade_moat
[ partitions INTEGERLIST ]
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ model_pointer STRING ]
)
|
( local_evidence ALIAS nond_local_evidence
[ sqp
| nip ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ model_pointer STRING ]
)
|
( local_interval_est ALIAS nond_local_interval_est
[ sqp
| nip ]

```

```

    [ convergence_tolerance REAL ]
    [ model_pointer STRING ]
  )
|
( local_reliability ALIAS nond_local_reliability
  [ mpp_search
    x_taylor_mean
    | u_taylor_mean
    | x_taylor_mpp
    | u_taylor_mpp
    | x_two_point
    | u_two_point
    | x_multi_point
    | u_multi_point
    | no_approx
    [ sqp
    | nip ]
    [ integration
      first_order
      | second_order
    [ probability_refinement ALIAS sample_refinement
    import
    | adapt_import
    | mm_adapt_import
    [ refinement_samples INTEGERLIST ]
    [ seed INTEGER > 0 ]
    ]
  ]
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
    series
    | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ final_moments
  none
  | standard
  | central
]
[ model_pointer STRING ]
)
|
( global_reliability ALIAS nond_global_reliability
  [ initial_samples INTEGER ]
  x_gaussian_process ALIAS x_kriging
  | u_gaussian_process ALIAS u_kriging
  [ ( surfpack
    [ export_model
    [ filename_prefix STRING ]
    ( formats
    [ text_archive ]

```

```

        [ binary_archive ]
      )
    ]
  )
| dakota
|
( experimental
  [ export_model
    [ filename_prefix STRING ]
    ( formats
      [ text_archive ]
      [ binary_archive ]
    )
  ]
  [ options_file STRING ]
]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ use_derivatives ]
[ seed INTEGER > 0 ]
[ rng
  mt19937
  | rnum2
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
    series
    | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ model_pointer STRING ]
)
|
( fsu_quasi_mc
  halton
  | hammersley
  [ latinize ]
  [ quality_metrics ]
  [ variance_based_decomp

```

```

    [ drop_tolerance REAL ]
  ]
  [ samples INTEGER ]
  [ fixed_sequence ]
  [ sequence_start INTEGERLIST ]
  [ sequence_leap INTEGERLIST ]
  [ prime_base INTEGERLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ model_pointer STRING ]
)
|
( vector_parameter_study
  final_point REALLIST
  | step_vector REALLIST
  num_steps INTEGER
  [ model_pointer STRING ]
)
|
( list_parameter_study
  list_of_points REALLIST
  |
  ( import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  )
  [ model_pointer STRING ]
)
|
( centered_parameter_study
  step_vector REALLIST
  steps_per_variable ALIAS deltas_per_variable INTEGERLIST
  [ model_pointer STRING ]
)
|
( multidim_parameter_study
  partitions INTEGERLIST
  [ model_pointer STRING ]
)
|
( richardson_extrap
  estimate_order
  | converge_order
  | converge_qoi
  [ refinement_rate REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ model_pointer STRING ]
)

```

KEYWORD model

```

  [ id_model STRING ]
  ( single ALIAS simulation
    [ interface_pointer STRING ]
    [ solution_level_control STRING ]
    [ solution_level_cost REALLIST
    | cost_recovery_metadata STRING ]
  )
  |
  ( surrogate
    [ id_surrogates INTEGERLIST ]
    ( global
      ( experimental_gaussian_process
        [ trend
        none
        | constant
        | linear
        | reduced_quadratic

```



```

| quadratic
]
[ num_restarts INTEGER > 1 ]
[ nugget REAL > 0
| find_nugget INTEGER ]
[ options_file STRING ]
[ export_approx_variance_file STRING
( ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
| annotated
| freeform ]
]
[ export_model
[ filename_prefix STRING ]
( formats
[ text_archive ]
[ binary_archive ]
)
]
[ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( gaussian_process ALIAS kriging
( dakota
[ point_selection ]
[ trend
constant
| linear
| reduced_quadratic
]
)
|
( surfpack
[ trend
constant
| linear
| reduced_quadratic
| quadratic
]
[ optimization_method STRING ]
[ max_trials INTEGER > 0 ]
[ nugget REAL > 0
| find_nugget INTEGER ]
[ correlation_lengths REALLIST ]
[ export_model
[ filename_prefix STRING ]
( formats
[ text_archive ]
[ binary_archive ]
[ algebraic_file ]
[ algebraic_console ]
)
]
[ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
[ export_approx_variance_file STRING
( ( custom_annotated
[ header ]
[ eval_id ]
[ interface_id ]
)
| annotated

```

```

| freeform ]
]
)
|
( mars
  [ max_bases INTEGER ]
  [ interpolation
linear
| cubic
]
  [ export_model
[ filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
)
]
  [ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( moving_least_squares
  [ basis_order ALIAS poly_order INTEGER >= 0 ]
  [ weight_function INTEGER ]
  [ export_model
[filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
)
]
  [ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( function_train
  [ regression_type
ls
|
( rls2
  l2_penalty REAL
)
]
  [ max_solver_iterations INTEGER >= 0 ]
  [ max_cross_iterations INTEGER >= 0 ]
  [ solver_tolerance REAL ]
  [ response_scaling ]
  [ tensor_grid ]
  [ rounding_tolerance REAL ]
  [ arithmetic_tolerance REAL ]
  [ start_order ALIAS order INTEGER >= 0 ]
[ dimension_preference REALLIST ]
]
  [ adapt_order ]
  [ kick_order INTEGER > 0 ]
  [ max_order INTEGER >= 0 ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  [ start_rank ALIAS rank INTEGER >= 0 ]
  [ adapt_rank ]
  [ kick_rank INTEGER > 0 ]
  [ max_rank INTEGER >= 0 ]
  [ max_cv_rank_candidates INTEGER >= 0 ]
)
|
( neural_network
  [ max_nodes ALIAS nodes INTEGER ]
  [ range REAL ]

```

```

    [ random_weight INTEGER ]
  [ export_model
[ filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
  [ algebraic_file ]
  [ algebraic_console ]
)
]
  [ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( radial_basis
  [ bases INTEGER ]
  [ max_pts INTEGER ]
  [ min_partition INTEGER ]
  [ max_subsets INTEGER ]
  [ export_model
[ filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
  [ algebraic_file ]
  [ algebraic_console ]
)
]
  [ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( polynomial
  basis_order INTEGER >= 0
  | linear
  | quadratic
  | cubic
  [ export_model
[ filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
  [ algebraic_file ]
  [ algebraic_console ]
)
]
  [ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( experimental_polynomial
  basis_order INTEGER >= 0
  [ options_file STRING ]
  [ export_model
[ filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
)
]
  [ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive

```

```

]
)
[ domain_decomposition
  [ cell_type STRING ]
  [ support_layers INTEGER ]
  [ discontinuity_detection
jump_threshold REAL
| gradient_threshold REAL
]
]
[ total_points INTEGER
| minimum_points
| recommended_points ]
[ ( dace_method_pointer STRING
[ auto_refinement
  [ max_iterations INTEGER > 0 ]
  [ max_function_evaluations INTEGER > 0 ]
  [ convergence_tolerance REAL ]
  [ soft_convergence_limit INTEGER >= 0 ]
  [ cross_validation_metric STRING
    [ folds INTEGER > 0 ]
  ]
]
]
)
| truth_model_pointer ALIAS actual_model_pointer STRING ]
[ reuse_points ALIAS reuse_samples
  all
  | region
  | none
]
[ import_build_points_file ALIAS import_points_file ALIAS samples_file STRING
  [ ( custom_annotated
  [ header
    [ use_variable_labels ]
  ]
  [ eval_id ]
  [ interface_id ]
  )
  |
  ( annotated
[ use_variable_labels ]
]
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ use_derivatives ]
[ correction
  zeroth_order
  | first_order
  | second_order
  additive
  | multiplicative
  | combined
]
[ metrics ALIAS diagnostics STRINGLIST
  [ cross_validation
[ folds INTEGER
| percent REAL ]
]
  [ press ]
]
[ import_challenge_points_file ALIAS challenge_points_file STRING
  [ ( custom_annotated
  [ header

```

```

        [ use_variable_labels ]
      ]
    [ eval_id ]
    [ interface_id ]
  )
  |
  ( annotated
  [ use_variable_labels ]
  ]
  | freeform ]
  [ active_only ]
  ]
)
|
( multipoint
  tana
  | qmea
  truth_model_pointer ALIAS actual_model_pointer STRING
)
|
( local
  taylor_series
  truth_model_pointer ALIAS actual_model_pointer STRING
)
|
( hierarchical
  ordered_model_fidelities ALIAS model_fidelity_sequence STRINGLIST
  [ correction
    zeroth_order
    | first_order
    | second_order
    additive
    | multiplicative
    | combined
  ]
)
|
( non_hierarchical ALIAS model_ensemble
  truth_model_pointer ALIAS actual_model_pointer STRING
  [ unordered_model_fidelities ALIAS approximation_models STRINGLIST ]
)
)
|
( nested
  [ optional_interface_pointer STRING
    [ optional_interface_responses_pointer STRING ]
  ]
  ( sub_method_pointer STRING
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
    [ primary_variable_mapping STRINGLIST ]
    [ secondary_variable_mapping STRINGLIST ]
    [ primary_response_mapping REALLIST ]
    [ secondary_response_mapping REALLIST ]
    [ identity_response_mapping ]
  )
)
|
( active_subspace ALIAS subspace
  truth_model_pointer ALIAS actual_model_pointer STRING
  [ initial_samples INTEGER ]
  [ sample_type
    lhs
    | random
  ]
  [ truncation_method
    [ bing_li ]
    [ constantine ]
    [ energy

```

```

    [ truncation_tolerance REAL ]
  ]
  [ cross_validation
    [ minimum
      | relative
      | decrease ]
    [ relative_tolerance REAL ]
    [ decrease_tolerance REAL ]
    [ max_rank INTEGER ]
    [ exhaustive ]
  ]
]
[ dimension INTEGER ]
[ bootstrap_samples INTEGER ]
[ build_surrogate
  [ refinement_samples INTEGERLIST ]
]
[ normalization
  mean_value
  | mean_gradient
  | local_gradient
]
)
|
( adapted_basis
  ( truth_model_pointer ALIAS actual_model_pointer STRING
    sparse_grid_level INTEGER
    |
    ( expansion_order INTEGER
      collocation_ratio REAL
    )
    [ dimension INTEGER ]
    [ rotation_method
      unranked
      | ranked
    ]
  )
  [ truncation_tolerance REAL ]
)
|
( random_field
  [ build_source
    rf_data_file STRING
    | dace_method_pointer STRING
    |
    ( analytic_covariance
      squared_exponential
      | exponential
    )
  ]
  [ expansion_form
    karhunen_loeve
    | principal_components
  ]
  [ expansion_bases INTEGER ]
  [ truncation_tolerance REAL ]
  propagation_model_pointer STRING
)
[ variables_pointer STRING ]
[ responses_pointer STRING ]
[ hierarchical_tagging ]

```

KEYWORD variables

```

[ id_variables STRING ]
[ active
  all
  | design
  | uncertain
  | aleatory
  | epistemic
  | state
]
[ mixed

```

```

| relaxed ]
[ continuous_design INTEGER > 0
  [ initial_point ALIAS cdv_initial_point REALLIST ]
  [ lower_bounds ALIAS cdv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS cdv_upper_bounds REALLIST ]
  [ scale_types ALIAS cdv_scale_types STRINGLIST ]
  [ scales ALIAS cdv_scales REALLIST ]
  [ descriptors ALIAS cdv_descriptors STRINGLIST ]
]
[ discrete_design_range INTEGER > 0
  [ initial_point ALIAS ddv_initial_point INTEGERLIST ]
  [ lower_bounds ALIAS ddv_lower_bounds INTEGERLIST ]
  [ upper_bounds ALIAS ddv_upper_bounds INTEGERLIST ]
  [ descriptors ALIAS ddv_descriptors STRINGLIST ]
]
[ discrete_design_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ categorical STRINGLIST
      [ adjacency_matrix INTEGERLIST ]
    ]
    [ initial_point INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ string INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values STRINGLIST
    [ adjacency_matrix INTEGERLIST ]
    [ initial_point STRINGLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ real INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values REALLIST
    [ categorical STRINGLIST
      [ adjacency_matrix INTEGERLIST ]
    ]
    [ initial_point REALLIST ]
    [ descriptors STRINGLIST ]
  ]
]
[ normal_uncertain INTEGER > 0
  means ALIAS nuv_means REALLIST
  std_deviations ALIAS nuv_std_deviations REALLIST
  [ lower_bounds ALIAS nuv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS nuv_upper_bounds REALLIST ]
  [ initial_point REALLIST ]
  [ descriptors ALIAS nuv_descriptors STRINGLIST ]
]
[ lognormal_uncertain INTEGER > 0
  ( lambdas ALIAS lnuv_lambdas REALLIST
    zetas ALIAS lnuv_zetas REALLIST
  )
  |
  ( means ALIAS lnuv_means REALLIST
    std_deviations ALIAS lnuv_std_deviations REALLIST
    | error_factors ALIAS lnuv_error_factors REALLIST
  )
  [ lower_bounds ALIAS lnuv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS lnuv_upper_bounds REALLIST ]
  [ initial_point REALLIST ]
  [ descriptors ALIAS lnuv_descriptors STRINGLIST ]
]
[ uniform_uncertain INTEGER > 0
  lower_bounds ALIAS uuv_lower_bounds REALLIST
  upper_bounds ALIAS uuv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS uuv_descriptors STRINGLIST ]
]
[ loguniform_uncertain INTEGER > 0
  lower_bounds ALIAS luuv_lower_bounds REALLIST
  upper_bounds ALIAS luuv_upper_bounds REALLIST

```

```

[ initial_point REALLIST ]
[ descriptors ALIAS luuv_descriptors STRINGLIST ]
]
[ triangular_uncertain INTEGER > 0
modes ALIAS tuv_modes REALLIST
lower_bounds ALIAS tuv_lower_bounds REALLIST
upper_bounds ALIAS tuv_upper_bounds REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS tuv_descriptors STRINGLIST ]
]
[ exponential_uncertain INTEGER > 0
betas ALIAS euv_betas REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS euv_descriptors STRINGLIST ]
]
[ beta_uncertain INTEGER > 0
alphas ALIAS buv_alphas REALLIST
betas ALIAS buv_betas REALLIST
lower_bounds ALIAS buv_lower_bounds REALLIST
upper_bounds ALIAS buv_upper_bounds REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS buv_descriptors STRINGLIST ]
]
[ gamma_uncertain INTEGER > 0
alphas ALIAS gauv_alphas REALLIST
betas ALIAS gauv_betas REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS gauv_descriptors STRINGLIST ]
]
[ gumbel_uncertain INTEGER > 0
alphas ALIAS guuv_alphas REALLIST
betas ALIAS guuv_betas REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS guuv_descriptors STRINGLIST ]
]
[ frechet_uncertain INTEGER > 0
alphas ALIAS fuv_alphas REALLIST
betas ALIAS fuv_betas REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS fuv_descriptors STRINGLIST ]
]
[ weibull_uncertain INTEGER > 0
alphas ALIAS wuv_alphas REALLIST
betas ALIAS wuv_betas REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS wuv_descriptors STRINGLIST ]
]
[ histogram_bin_uncertain INTEGER > 0
[ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
abscissas ALIAS huv_bin_abscissas REALLIST
ordinates ALIAS huv_bin_ordinates REALLIST
| counts ALIAS huv_bin_counts REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS huv_bin_descriptors STRINGLIST ]
]
[ poisson_uncertain INTEGER > 0
lambdas REALLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ binomial_uncertain INTEGER > 0
probability_per_trial ALIAS prob_per_trial REALLIST
num_trials INTEGERLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ negative_binomial_uncertain INTEGER > 0
probability_per_trial ALIAS prob_per_trial REALLIST
num_trials INTEGERLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ geometric_uncertain INTEGER > 0

```



```

probability_per_trial ALIAS prob_per_trial REALLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ hypergeometric_uncertain INTEGER > 0
total_population INTEGERLIST
selected_population INTEGERLIST
num_drawn INTEGERLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ histogram_point_uncertain
[ integer INTEGER > 0
[ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
abscissas INTEGERLIST
counts REALLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ string INTEGER > 0
[ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
abscissas STRINGLIST
counts REALLIST
[ initial_point STRINGLIST ]
[ descriptors STRINGLIST ]
]
[ real INTEGER > 0
[ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
abscissas REALLIST
counts REALLIST
[ initial_point REALLIST ]
[ descriptors STRINGLIST ]
]
]
[ uncertain_correlation_matrix REALLIST ]
[ continuous_interval_uncertain ALIAS interval_uncertain INTEGER > 0
[ num_intervals ALIAS iuv_num_intervals INTEGERLIST ]
[ interval_probabilities ALIAS interval_probs ALIAS iuv_interval_probs REALLIST ]
lower_bounds REALLIST
upper_bounds REALLIST
[ initial_point REALLIST ]
[ descriptors ALIAS iuv_descriptors STRINGLIST ]
]
[ discrete_interval_uncertain INTEGER > 0
[ num_intervals INTEGERLIST ]
[ interval_probabilities ALIAS interval_probs ALIAS range_probabilities ALIAS range_probs REALLIST ]
lower_bounds INTEGERLIST
upper_bounds INTEGERLIST
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ discrete_uncertain_set
[ integer INTEGER > 0
[ elements_per_variable ALIAS num_set_values INTEGERLIST ]
elements ALIAS set_values INTEGERLIST
[ set_probabilities ALIAS set_probs REALLIST ]
[ categorical STRINGLIST ]
[ initial_point INTEGERLIST ]
[ descriptors STRINGLIST ]
]
[ string INTEGER > 0
[ elements_per_variable ALIAS num_set_values INTEGERLIST ]
elements ALIAS set_values STRINGLIST
[ set_probabilities ALIAS set_probs REALLIST ]
[ initial_point STRINGLIST ]
[ descriptors STRINGLIST ]
]
[ real INTEGER > 0
[ elements_per_variable ALIAS num_set_values INTEGERLIST ]
elements ALIAS set_values REALLIST
[ set_probabilities ALIAS set_probs REALLIST ]
[ categorical STRINGLIST ]
[ initial_point REALLIST ]
]
]

```

```

    [ descriptors STRINGLIST ]
  ]
]
[ continuous_state INTEGER > 0
  [ initial_state ALIAS csv_initial_state REALLIST ]
  [ lower_bounds ALIAS csv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS csv_upper_bounds REALLIST ]
  [ descriptors ALIAS csv_descriptors STRINGLIST ]
]
[ discrete_state_range INTEGER > 0
  [ initial_state ALIAS dsv_initial_state INTEGERLIST ]
  [ lower_bounds ALIAS dsv_lower_bounds INTEGERLIST ]
  [ upper_bounds ALIAS dsv_upper_bounds INTEGERLIST ]
  [ descriptors ALIAS dsv_descriptors STRINGLIST ]
]
[ discrete_state_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ categorical STRINGLIST ]
    [ initial_state INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ string INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values STRINGLIST
    [ initial_state STRINGLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ real INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values REALLIST
    [ categorical STRINGLIST ]
    [ initial_state REALLIST ]
    [ descriptors STRINGLIST ]
  ]
]
[ linear_inequality_constraint_matrix REALLIST ]
[ linear_inequality_lower_bounds REALLIST ]
[ linear_inequality_upper_bounds REALLIST ]
[ linear_inequality_scale_types STRINGLIST ]
[ linear_inequality_scales REALLIST ]
[ linear_equality_constraint_matrix REALLIST ]
[ linear_equality_targets REALLIST ]
[ linear_equality_scale_types STRINGLIST ]
[ linear_equality_scales REALLIST ]

```

KEYWORD interface

```

[ id_interface STRING ]
[ analysis_drivers STRINGLIST
  [ input_filter STRING ]
  [ output_filter STRING ]
  ( system
    [ parameters_file STRING ]
    [ results_file STRING ]
    [ file_tag ]
    [ file_save ]
    [ labeled ]
    [ aprepro ALIAS dprepro ]
    [ work_directory
      [ named STRING ]
      [ directory_tag ALIAS dir_tag ]
      [ directory_save ALIAS dir_save ]
      [ link_files STRINGLIST ]
      [ copy_files STRINGLIST ]
      [ replace ]
    ]
    [ allow_existing_results ]
    [ verbatim ]
  )
]
( fork
  [ parameters_file STRING ]

```

```

    [ results_file STRING ]
    [ file_tag ]
    [ file_save ]
    [ labeled ]
    [ aprepro ALIAS dprepro ]
    [ work_directory
      [ named STRING ]
      [ directory_tag ALIAS dir_tag ]
      [ directory_save ALIAS dir_save ]
      [ link_files STRINGLIST ]
      [ copy_files STRINGLIST ]
      [ replace ]
    ]
    [ allow_existing_results ]
    [ verbatim ]
  )
|
( direct
  [ processors_per_analysis INTEGER > 0 ]
)
|
( plugin
  library_path STRING
)
| matlab
|
( python
  [ numpy ]
)
|
( legacy_python
  [ numpy ]
)
| scilab
| grid
[ analysis_components STRINGLIST ]
]
[ algebraic_mappings STRING ]
[ failure_capture
  abort
  | retry INTEGER
  | recover REALLIST
  | continuation
]
[ deactivate
  [ active_set_vector ]
  [ evaluation_cache ]
  [ strict_cache_equality
    [ cache_tolerance REAL ]
  ]
  [ restart_file ]
]
|
[ ( batch
  [ size INTEGER > 0 ]
)
|
( asynchronous
  [ evaluation_concurrency INTEGER > 0 ]
  [ local_evaluation_scheduling
    dynamic
    | static
  ]
  [ analysis_concurrency INTEGER > 0 ]
]
[ evaluation_servers INTEGER > 0 ]
[ evaluation_scheduling
  master
  |
  ( peer
    dynamic
    | static
  )
]
]

```

```
[ processors_per_evaluation INTEGER > 0 ]
[ analysis_servers INTEGER > 0 ]
[ analysis_scheduling
  master
  | peer
  ]
```

KEYWORD responses

```
[ id_responses STRING ]
[ descriptors ALIAS response_descriptors STRINGLIST ]
( objective_functions ALIAS num_objective_functions INTEGER >= 0
  [ sense STRINGLIST ]
  [ primary_scale_types ALIAS objective_function_scale_types STRINGLIST ]
  [ primary_scales ALIAS objective_function_scales REALLIST ]
  [ weights ALIAS multi_objective_weights REALLIST ]
  [ nonlinear_inequality_constraints ALIAS num_nonlinear_inequality_constraints INTEGER >= 0
    [ lower_bounds ALIAS nonlinear_inequality_lower_bounds REALLIST ]
    [ upper_bounds ALIAS nonlinear_inequality_upper_bounds REALLIST ]
    [ scale_types ALIAS nonlinear_inequality_scale_types STRINGLIST ]
    [ scales ALIAS nonlinear_inequality_scales REALLIST ]
  ]
  [ nonlinear_equality_constraints ALIAS num_nonlinear_equality_constraints INTEGER >= 0
    [ targets ALIAS nonlinear_equality_targets REALLIST ]
    [ scale_types ALIAS nonlinear_equality_scale_types STRINGLIST ]
    [ scales ALIAS nonlinear_equality_scales REALLIST ]
  ]
  [ scalar_objectives ALIAS num_scalar_objectives INTEGER >= 0 ]
  [ field_objectives ALIAS num_field_objectives INTEGER >= 0
    lengths INTEGERLIST
    [ num_coordinates_per_field INTEGERLIST ]
    [ read_field_coordinates ]
  ]
)
|
( calibration_terms ALIAS least_squares_terms ALIAS num_least_squares_terms INTEGER >= 0
  [ scalar_calibration_terms INTEGER >= 0 ]
  [ field_calibration_terms INTEGER >= 0
    lengths INTEGERLIST
    [ num_coordinates_per_field INTEGERLIST ]
    [ read_field_coordinates ]
  ]
  [ primary_scales ALIAS calibration_term_scales ALIAS least_squares_term_scales REALLIST ]
  [ weights ALIAS calibration_weights ALIAS least_squares_weights REALLIST ]
  [ ( calibration_data
    [ data_directory STRING ]
    [ num_experiments INTEGER >= 0 ]
    [ num_config_variables INTEGER >= 0 ]
    [ experiment_variance_type ALIAS variance_type STRINGLIST ]
    [ scalar_data_file STRING
      [ ( custom_annotated
        [ header ]
        [ exp_id ]
        )
      | annotated
      | freeform ]
    ]
    [ interpolate ]
  )
  |
  ( calibration_data_file ALIAS least_squares_data_file STRING
    [ ( custom_annotated
      [ header ]
      [ exp_id ]
      )
    | annotated
    | freeform ]
    [ num_experiments INTEGER >= 0 ]
    [ num_config_variables INTEGER >= 0 ]
    [ experiment_variance_type ALIAS variance_type STRINGLIST ]
  ]
  [ simulation_variance REALLIST ]
  [ nonlinear_inequality_constraints ALIAS num_nonlinear_inequality_constraints INTEGER >= 0
    [ lower_bounds ALIAS nonlinear_inequality_lower_bounds REALLIST ]
```

```

    [ upper_bounds ALIAS nonlinear_inequality_upper_bounds REALLIST ]
    [ scale_types ALIAS nonlinear_inequality_scale_types STRINGLIST ]
    [ scales ALIAS nonlinear_inequality_scales REALLIST ]
  ]
  [ nonlinear_equality_constraints ALIAS num_nonlinear_equality_constraints INTEGER >= 0
    [ targets ALIAS nonlinear_equality_targets REALLIST ]
    [ scale_types ALIAS nonlinear_equality_scale_types STRINGLIST ]
    [ scales ALIAS nonlinear_equality_scales REALLIST ]
  ]
)
|
( response_functions ALIAS num_response_functions INTEGER >= 0
  [ scalar_responses ALIAS num_scalar_responses INTEGER >= 0 ]
  [ field_responses ALIAS num_field_responses INTEGER >= 0
    lengths INTEGERLIST
    [ num_coordinates_per_field INTEGERLIST ]
    [ read_field_coordinates ]
  ]
)
no_gradients
| analytic_gradients
|
( mixed_gradients
  id_numerical_gradients INTEGERLIST
  id_analytic_gradients INTEGERLIST
  [ method_source ]
  [ ( dakota
    [ ignore_bounds ]
    [ relative
      | absolute
      | bounds ]
    )
  | vendor ]
  [ interval_type ]
  [ forward
  | central ]
  [ fd_step_size ALIAS fd_gradient_step_size REALLIST ]
)
|
( numerical_gradients
  [ method_source ]
  [ ( dakota
    [ ignore_bounds ]
    [ relative
      | absolute
      | bounds ]
    )
  | vendor ]
  [ interval_type ]
  [ forward
  | central ]
  [ fd_step_size ALIAS fd_gradient_step_size REALLIST ]
)
no_hessians
|
( numerical_hessians
  [ fd_step_size ALIAS fd_hessian_step_size REALLIST ]
  [ relative
  | absolute
  | bounds ]
  [ forward
  | central ]
)
|
( quasi_hessians
  ( bfgs
    [ damped ]
  )
  | srl
)
| analytic_hessians
|
( mixed_hessians

```

```
[ id_numerical_hessians INTEGERLIST
  [ fd_step_size ALIAS fd_hessian_step_size REALLIST ]
]
[ relative
| absolute
| bounds ]
[ forward
| central ]
[ id_quasi_hessians INTEGERLIST
  ( bfgs
    [ damped ]
  )
| srl
]
[ id_analytic_hessians INTEGERLIST ]
)
[ metadata STRINGLIST ]
```

Chapter 6

Topics Area

This page introduces the user to the topics used to organize keywords.

- [admin](#)
- [dakota_IO](#)
- [dakota_concepts](#)
- [models](#)
- [variables](#)
- [responses](#)
- [interface](#)
- [methods](#)
- [advanced_topics](#)
- [packages](#)

6.1 admin

Description

This is only for management while ref man is under construction

Related Topics

- [empty](#)
- [problem](#)
- [not_yet_reviewed](#)

Related Keywords

6.1.1 empty

Description

This topic tracks the keywords which do not have content in the reference manual

Related Topics

Related Keywords

6.1.2 problem

Description

empty

Related Topics

Related Keywords

6.1.3 not_yet_reviewed

Description

Not yet reviewed.

Related Topics

Related Keywords

6.2 dakota_IO

Description

Keywords and Concepts relating inputs to Dakota and outputs from Dakota

Related Topics

- [dakota_inputs](#)
- [dakota_output](#)
- [file_formats](#)

Related Keywords

- [error_file](#) : Base filename for error redirection
- [output_file](#) : Base filename for output redirection
- [input](#) : Base filename for post-run mode data input
- [output](#) : Base filename for post-run mode data output
- [input](#) : Base filename for pre-run mode data input
- [output](#) : Base filename for pre-run mode data output
- [read_restart](#) : Base filename for restart file read
- [stop_restart](#) : Restart record number at which to stop reading the restart file.

- [input](#) : Base filename for run mode data input
- [output](#) : Base filename for run mode data output
- [write_restart](#) : Base filename for restart file write

6.2.1 dakota_inputs

Description

empty

Related Topics

- [block](#)
- [data_import_capabilities](#)

Related Keywords

6.2.1.1 block

Description

A block is the highest level of keyword organization in Dakota. There are currently 6 blocks in the Dakota input spec:

Related Topics

- [block_identifier](#)
- [block_pointer](#)

Related Keywords

- [environment](#) : Top-level settings for Dakota execution
- [interface](#) : Specifies how function evaluations will be performed in order to map the variables into the responses.
- [method](#) : Begins Dakota method selection and behavioral settings.
- [model](#) : Specifies how variables are mapped into a set of responses
- [responses](#) : Description of the model output data returned to Dakota upon evaluation of an interface.
- [variables](#) : Specifies the parameter set to be iterated by a particular method.

6.2.1.2 block_identifier

Description

empty

Related Topics

Related Keywords

- [id_interface](#) : Name the interface block; helpful when there are multiple
- [id_method](#) : Name the method block; helpful when there are multiple
- [id_model](#) : Give the model block an identifying name, in case of multiple model blocks
- [id_responses](#) : Name the responses block; helpful when there are multiple
- [id_variables](#) : Name the variables block; helpful when there are multiple

6.2.1.3 block_pointer

Description

See [block_pointer](#) for details about pointers.

Related Topics

Related Keywords

- [top_method_pointer](#) : Identify which method leads the Dakota study
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method

- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer_list` : Associate models with method names
- `method_pointer_list` : Pointers to methods to execute sequentially or collaboratively
- `global_model_pointer` : Pointer to model used by global method
- `global_method_pointer` : Pointer to global method
- `local_model_pointer` : Pointer to model used by local method
- `local_method_pointer` : Pointer to local method
- `model_pointer_list` : Associate models with method names
- `method_pointer_list` : Pointers to methods to execute sequentially or collaboratively
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `method_pointer` : Pointer to sub-method to run from each starting point
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method

- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to optimization or least-squares sub-method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [truth_model_pointer](#) : Pointer to specify a full-space model, from which to construct a lower dimensional surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate

- [optional_interface_pointer](#) : Pointer to interface that provides non-nested responses
- [optional_interface_responses_pointer](#) : Pointer to responses block that defines non-nested responses
- [sub_method_pointer](#) : The `sub_method_pointer` specifies the method block for the sub-iterator
- [responses_pointer](#) : Specify which responses block will be used by this model block
- [interface_pointer](#) : Interface block pointer for the single model type
- [dace_method_pointer](#) : Specify a method to gather training data
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [variables_pointer](#) : Specify which variables block will be included with this model block

6.2.1.4 data_import_capabilities

Description

empty

Related Topics

Related Keywords

6.2.2 dakota_output

Description

empty

Related Topics

Related Keywords

- [graphics](#) : (DEPRECATED) Display plots of variables and responses
- [output_precision](#) : Control the output precision
- [results_output](#) : (Experimental) Write a summary file containing the final results
- [hdf5](#) : Write results to file in HDF5 format
- [interface_selection](#) : Select the models that write evaluation data to HDF5
- [all](#) : Write evaluation data for all interfaces to HDF5
- [none](#) : Write evaluation data for no interfaces to HDF5
- [simulation](#) : Write evaluation data only for simulation interfaces to HDF5
- [model_selection](#) : Select the models that write evaluation data to HDF5
- [all](#) : Write evaluation data to HDF5 for all models
- [all_methods](#) : Write evaluation data to HDF5 for all models that belong directly to methods

- [none](#) : Write evaluation data for no models to HDF5
- [top_method](#) : Write evaluation data only for the top-level method's model to HDF5
- [results_output_file](#) : The base file name of the results file
- [text](#) : Write results to file in text format
- [tabular_data](#) : Write a tabular results file with variable and response history
- [tabular_data_file](#) : File name for tabular data output
- [output](#) : Control how much method information is written to the screen and output file

6.2.3 file_formats

Description

See sections "Inputs to Dakota" and "Outputs from Dakota" in the Dakota User's Manual [5].

Related Topics

Related Keywords

- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [aprepro](#) : Write parameters files in APREPRO syntax
- [labeled](#) : Requires correct function value labels in results file
- [aprepro](#) : Write parameters files in APREPRO syntax
- [labeled](#) : Requires correct function value labels in results file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format

- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format

- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format

- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format

- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format

- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format

- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file

- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format

- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [active_only](#) : Import only active variables from tabular data file
- [annotated](#) : Selects annotated tabular file format
- [custom_annotated](#) : Selects custom-annotated tabular file format
- [freeform](#) : Selects freeform file format
- [annotated](#) : Selects annotated tabular file format for experiment data
- [custom_annotated](#) : Selects custom-annotated tabular file format for experiment data
- [freeform](#) : Selects free-form tabular file format for experiment data
- [annotated](#) : Selects annotated tabular file format for experiment data
- [custom_annotated](#) : Selects custom-annotated tabular file format for experiment data
- [freeform](#) : Selects free-form tabular file format for experiment data

6.3 dakota_concepts

Description

Miscellaneous concepts related to Dakota operation

Related Topics

- [method_independent_controls](#)
- [block](#)
- [strategies](#)
- [command_line_options](#)
- [restarts](#)
- [pointers](#)

Related Keywords

6.3.1 method_independent_controls

Description

The `<method independent controls>` are those controls which are valid for a variety of methods. In some cases, these controls are abstractions which may have slightly different implementations from one method to the next. While each of these controls is not valid for every method, the controls are valid for enough methods that it was reasonable to consolidate the specifications.

Related Topics

Related Keywords

- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [convergence_tolerance](#) : Stopping criterion based on relative error reduction
- [max_function_evaluations](#) : Stopping criterion based on maximum function evaluations
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods

- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence

- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [final_solutions](#) : Number of designs returned as the best solutions
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers

- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [id_method](#) : Name the method block; helpful when there are multiple
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [convergence_tolerance](#) : Stopping criterion based on relative error reduction
- [max_function_evaluations](#) : Stopping criterion based on maximum function evaluations
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [convergence_tolerance](#) : Stopping criterion based on relative error reduction
- [max_function_evaluations](#) : Stopping criterion based on maximum function evaluations
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [max_function_evaluations](#) : Stopping criterion based on maximum function evaluations
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers

- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence

- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [speculative](#) : Compute speculative gradients
- [output](#) : Control how much method information is written to the screen and output file
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [scaling](#) : Turn on scaling for variables, responses, and constraints
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [constraint_tolerance](#) : Maximum allowable constraint violation still considered feasible
- [convergence_tolerance](#) : Stopping criterion based on objective function or statistics convergence
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods
- [max_function_evaluations](#) : Number of function evaluations allowed for optimizers
- [max_iterations](#) : Number of iterations allowed for optimizers and adaptive UQ methods

6.3.2 block

Description

A block is the highest level of keyword organization in Dakota. There are currently 6 blocks in the Dakota input spec:

Related Topics

- [block_identifier](#)
- [block_pointer](#)

Related Keywords

- [environment](#) : Top-level settings for Dakota execution
- [interface](#) : Specifies how function evaluations will be performed in order to map the variables into the responses.
- [method](#) : Begins Dakota method selection and behavioral settings.
- [model](#) : Specifies how variables are mapped into a set of responses
- [responses](#) : Description of the model output data returned to Dakota upon evaluation of an interface.
- [variables](#) : Specifies the parameter set to be iterated by a particular method.

6.3.2.1 block_identifier

Description

empty

Related Topics

Related Keywords

- [id_interface](#) : Name the interface block; helpful when there are multiple
- [id_method](#) : Name the method block; helpful when there are multiple
- [id_model](#) : Give the model block an identifying name, in case of multiple model blocks
- [id_responses](#) : Name the responses block; helpful when there are multiple
- [id_variables](#) : Name the variables block; helpful when there are multiple

6.3.2.2 block_pointer

Description

See [block_pointer](#) for details about pointers.

Related Topics

Related Keywords

- [top_method_pointer](#) : Identify which method leads the Dakota study
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method

- [model_pointer_list](#) : Associate models with method names
- [method_pointer_list](#) : Pointers to methods to execute sequentially or collaboratively
- [global_model_pointer](#) : Pointer to model used by global method
- [global_method_pointer](#) : Pointer to global method
- [local_model_pointer](#) : Pointer to model used by local method
- [local_method_pointer](#) : Pointer to local method
- [model_pointer_list](#) : Associate models with method names
- [method_pointer_list](#) : Pointers to methods to execute sequentially or collaboratively
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to run from each starting point
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method

- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to optimization or least-squares sub-method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [truth_model_pointer](#) : Pointer to specify a full-space model, from which to construct a lower dimensional surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [optional_interface_pointer](#) : Pointer to interface that provides non-nested responses
- [optional_interface_responses_pointer](#) : Pointer to responses block that defines non-nested responses
- [sub_method_pointer](#) : The `sub_method_pointer` specifies the method block for the sub-iterator
- [responses_pointer](#) : Specify which responses block will be used by this model block
- [interface_pointer](#) : Interface block pointer for the single model type
- [dace_method_pointer](#) : Specify a method to gather training data
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [variables_pointer](#) : Specify which variables block will be included with this model block

6.3.3 strategies

Description

empty

Related Topics

- [advanced_strategies](#)

Related Keywords

6.3.3.1 advanced_strategies

Description

empty

Related Topics

Related Keywords

6.3.4 command_line_options

Description

empty

Related Topics

Related Keywords

- [check](#) : Invoke Dakota in input check mode
- [error_file](#) : Base filename for error redirection
- [output_file](#) : Base filename for output redirection
- [post_run](#) : Invoke Dakota with post-run mode active
- [pre_run](#) : Invoke Dakota with pre-run mode active
- [read_restart](#) : Base filename for restart file read
- [run](#) : Invoke Dakota with run mode active
- [write_restart](#) : Base filename for restart file write

6.3.5 restarts

Description

empty

Related Topics

Related Keywords

6.3.6 pointers

Description

For all pointer specifications, if a pointer string is specified and no corresponding id string is available, Dakota will exit with an error message.

If the pointer is optional and no pointer string is specified, then the last specification parsed will be used.

It is appropriate to omit optional cross-referencing whenever the relationships are unambiguous due to the presence of only one specification.

Related Topics

- [block_pointer](#)
- [objective_function_pointer](#)

Related Keywords

6.3.6.1 block_pointer

Description

See [block_pointer](#) for details about pointers.

Related Topics

Related Keywords

- [top_method_pointer](#) : Identify which method leads the Dakota study
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method

- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer_list` : Associate models with method names
- `method_pointer_list` : Pointers to methods to execute sequentially or collaboratively
- `global_model_pointer` : Pointer to model used by global method
- `global_method_pointer` : Pointer to global method
- `local_model_pointer` : Pointer to model used by local method
- `local_method_pointer` : Pointer to local method
- `model_pointer_list` : Associate models with method names
- `method_pointer_list` : Pointers to methods to execute sequentially or collaboratively
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `model_pointer` : Identifier for model block to be used by a method
- `method_pointer` : Pointer to sub-method to run from each starting point

- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to optimization or least-squares sub-method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method

- [method_pointer](#) : Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [model_pointer](#) : Identifier for model block to be used by a method
- [truth_model_pointer](#) : Pointer to specify a full-space model, from which to construct a lower dimensional surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [optional_interface_pointer](#) : Pointer to interface that provides non-nested responses
- [optional_interface_responses_pointer](#) : Pointer to responses block that defines non-nested responses
- [sub_method_pointer](#) : The `sub_method_pointer` specifies the method block for the sub-iterator
- [responses_pointer](#) : Specify which reponses block will be used by this model block
- [interface_pointer](#) : Interface block pointer for the single model type
- [dace_method_pointer](#) : Specify a method to gather training data
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [truth_model_pointer](#) : Pointer to specify a "truth" model, from which to construct a surrogate
- [variables_pointer](#) : Specify which variables block will be included with this model block

6.3.6.2 objective_function_pointer

Description

See [block_pointer](#) for details about pointers.

Related Topics

Related Keywords

- [id_analytic_gradients](#) : Identify which analytical gradient corresponds to which response
- [id_numerical_gradients](#) : Identify which numerical gradient corresponds to which response
- [id_analytic_hessians](#) : Identify which analytical Hessian corresponds to which response
- [id_numerical_hessians](#) : Identify which numerical-Hessian corresponds to which response
- [id_quasi_hessians](#) : Identify which quasi-Hessian corresponds to which response

6.4 models

Description

Keywords and Concepts relating to the `model` block

Related Topics

- [surrogate_models](#)
- [recast_models](#)
- [multifidelity_models](#)
- [reduced_order_models](#)
- [nested_models](#)
- [advanced_model_recursion](#)

Related Keywords

6.4.1 surrogate_models

Description

empty

Related Topics

- [surrogate_based_optimization_methods](#)

Related Keywords

- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export

- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [auto_refinement](#) : Experimental auto-refinement of surrogate model
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [point_selection](#) : Enable greedy selection of well-spaced build points
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [algebraic_console](#) : Export surrogate model in algebraic format to the console
- [algebraic_file](#) : Export surrogate model in algebraic format to a file
- [binary_archive](#) : Surrogate model binary archive file format

- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [metrics](#) : Compute surrogate quality metrics
- [cross_validation](#) : Perform k-fold cross validation
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [algebraic_console](#) : Export surrogate model in algebraic format to the console
- [algebraic_file](#) : Export surrogate model in algebraic format to a file
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [max_nodes](#) : Maximum number of hidden layer nodes
- [random_weight](#) : (Inactive) Random weight control
- [range](#) : Range for neural network random weights
- [export_model](#) : Exports surrogate model in user-specified format(s)

- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [algebraic_console](#) : Export surrogate model in algebraic format to the console
- [algebraic_file](#) : Export surrogate model in algebraic format to a file
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [bases](#) : Initial number of radial basis functions
- [export_model](#) : Exports surrogate model in user-specified format(s)
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [formats](#) : Formats for surrogate model export
- [algebraic_console](#) : Export surrogate model in algebraic format to the console
- [algebraic_file](#) : Export surrogate model in algebraic format to a file
- [binary_archive](#) : Surrogate model binary archive file format
- [text_archive](#) : Surrogate model plain-text archive file format
- [binary_archive](#) : Surrogate model binary archive file format
- [filename_prefix](#) : User-customizable portion of exported/imported surrogate model filenames
- [text_archive](#) : Surrogate model plain-text archive file format
- [max_pts](#) : Maximum number of RBF CVT points
- [max_subsets](#) : Number of trial RBF subsets
- [min_partition](#) : (Inactive) Minimum RBF partition
- [reuse_points](#) : Surrogate model training data reuse control

6.4.2 surrogate_based_optimization_methods

Description

empty

Related Topics

Related Keywords

- [efficient_global](#) : Global Surrogate Based Optimization, a.k.a. EGO
- [surrogate_based_global](#) : Adaptive Global Surrogate-Based Optimization
- [surrogate_based_local](#) : Local Surrogate Based Optimization

6.4.3 recast_models

Description

empty

Related Topics

Related Keywords

6.4.4 multifidelity_models

Description

empty

Related Topics

Related Keywords

6.4.5 reduced_order_models

Description

empty

Related Topics

Related Keywords

6.4.6 nested_models

Description

empty

Related Topics

Related Keywords

6.4.7 advanced_model_recursion

Description

empty

Related Topics

- [hybrid_and_recursions_logic](#)

Related Keywords

6.4.8 hybrid_and_recursions_logic

Description

empty

Related Topics

Related Keywords

6.5 variables

Description

Keywords and concepts relating to the `variables` block, which defines the parameters for the study.

Related Topics

- [variable_domain](#)
- [linear_constraints](#)
- [variable_type](#)

Related Keywords

6.5.1 variable_domain

Description

Dakota variables can be grouped by their valid domains.

1. Mixed: continuous and discrete variables are treated separately
2. Relaxed: noncategorical discrete variables are relaxed and treated as continuous variables (categorical variables are non-relaxable and remain discrete)

Refer to [mixed](#) and [relaxed](#) for additional information.

Related Topics

- [continuous_variables](#)
- [discrete_variables](#)

Related Keywords

6.5.2 continuous_variables

Description

This page collects information related to the topic of continuous design, uncertain, and state variables.

Related Topics

Related Keywords

- [beta_uncertain](#) : Aleatory uncertain variable - beta
- [continuous_design](#) : Design variable - continuous
- [continuous_interval_uncertain](#) : Epistemic uncertain variable - values from one or more continuous intervals
- [continuous_state](#) : State variable - continuous
- [exponential_uncertain](#) : Aleatory uncertain variable - exponential
- [frechet_uncertain](#) : Aleatory uncertain variable - Frechet
- [gamma_uncertain](#) : Aleatory uncertain variable - gamma
- [gumbel_uncertain](#) : Aleatory uncertain variable - gumbel
- [histogram_bin_uncertain](#) : Aleatory uncertain variable - continuous histogram
- [lognormal_uncertain](#) : Aleatory uncertain variable - lognormal
- [loguniform_uncertain](#) : Aleatory uncertain variable - loguniform
- [normal_uncertain](#) : Aleatory uncertain variable - normal (Gaussian)
- [triangular_uncertain](#) : Aleatory uncertain variable - triangular
- [uniform_uncertain](#) : Aleatory uncertain variable - uniform
- [weibull_uncertain](#) : Aleatory uncertain variable - Weibull

6.5.3 discrete_variables

Description

This page discusses discrete design, uncertain, and state variables (which have `discrete` in their keyword name) as they have similar specifications. These include:

1. Integer ranges
2. Sets of integers
3. Sets of reals
4. Sets of strings and each is described below.

In addition, some aleatory uncertain variables, e.g., [binomial_uncertain](#), are discrete integer-valued random variables specified using parameters. These are described on their individual keyword pages.

Sets

Sets of integers, reals, and strings have similar specifications, though different value types.

The variables are specified using three keywords:

- Variable declaration keyword - specifies the number of variables being defined
- `elements_per_variable` - a list of positive integers specifying how many set members each variable admits
 - Length = # of variables

- `elements` - a list of the permissible integer values in ALL sets, concatenated together.
 - Length = sum of `elements_per_variable`, or an integer multiple of number of variables
 - The order is very important here.
 - The list is partitioned according to the values of `elements_per_variable`, and each partition is assigned to a variable.
- The ordering of `elements_per_variable`, and the partitions of `elements` must match the strings from `descriptors`

For string variables, each string element value must be quoted and may contain alphanumeric, dash, underscore, and colon. White space, quote characters, and backslash/metacharacters are not permitted.

Examples are given on the pages:

- discrete design set [integer](#)
- discrete design set [real](#)
- discrete design set [string](#)
- discrete uncertain set [integer](#)
- discrete uncertain set [real](#)
- discrete uncertain set [string](#)

Range

For discrete variables defined by `range(s)`, the `lower_bounds` and `upper_bounds` restrict the permissible values. For design variables, this constrains the feasible design space and is frequently used to prevent nonphysical designs. This is a discrete interval variable that may take any integer value within bounds (e.g., [1, 4], allowing values of 1, 2, 3, or 4). For some variable types, each variable is can be defined by multiple ranges.

Examples are given on the pages:

- [discrete_interval_uncertain](#)

Related Topics

Related Keywords

- [binomial_uncertain](#) : Aleatory uncertain discrete variable - binomial
- [discrete_design_range](#) : Design variable - discrete range-valued
- [discrete_design_set](#) : Design variable - discrete set-valued
- [integer](#) : Integer-valued discrete design variables
- [real](#) : Real-valued discrete design variables
- [string](#) : String-valued discrete design set variables
- [discrete_interval_uncertain](#) : Epistemic uncertain variable - values from one or more discrete intervals
- [discrete_state_range](#) : State variables - discrete range-valued
- [discrete_state_set](#) : State variable - discrete set-valued
- [integer](#) : Discrete state variables, each defined by a set of permissible integers
- [real](#) : Discrete state variables, each defined by a set of permissible real numbers

- [string](#) : String-valued discrete state set variables
- [discrete_uncertain_set](#) : Epistemic uncertain variable - discrete set-valued
- [integer](#) : Discrete, epistemic uncertain variable - integers within a set
- [real](#) : Discrete, epistemic uncertain variable - real numbers within a set
- [string](#) : Discrete, epistemic uncertain variable - strings within a set
- [geometric_uncertain](#) : Aleatory uncertain discrete variable - geometric
- [histogram_point_uncertain](#) : Aleatory uncertain variable - discrete histogram
- [hypergeometric_uncertain](#) : Aleatory uncertain discrete variable - hypergeometric
- [negative_binomial_uncertain](#) : Aleatory uncertain discrete variable - negative binomial
- [poisson_uncertain](#) : Aleatory uncertain discrete variable - Poisson

6.5.4 linear_constraints

Description

Many methods can make use of linear equality or inequality constraints.

As the name implies, linear constraints are constraints that are linear functions of the variables. Constraints that are nonlinear functions of variables are specified using the [nonlinear_constraints](#) family of keywords. From a Dakota usage point of view, the most important difference between linear and nonlinear constraints is that the former are specified entirely within the Dakota input file and calculated by Dakota itself, while the latter must be calculated by the user's simulation and returned as responses to Dakota.

The Optimization chapter of the User's Manual[5] states which methods support linear constraints. Of those methods, a subset strictly obey linear constraints; that is, no candidate points are generated by the optimizer that violate the constraints. These include [asynch_pattern_search](#), the `optpp_*` family of optimizers (with the exception of `optpp_fd_newton`), and [npsol_sqp](#). The other methods seek feasible solutions (i.e. solutions that satisfy the linear constraints), but may violate the constraints as they run. Linear constraints may also be violated, even when using an optimizer that itself strictly respects them, if [numerical_gradients](#) are used. In this case, Dakota may request evaluations that lie outside of the feasible region when computing a gradient near the boundary.

One final limitation that bears mentioning is that linear constraints are compatible only with continuous variables. No discrete types are permitted when using linear constraints.

Related Topics

Related Keywords

- [linear_equality_constraint_matrix](#) : Define coefficients of the linear equalities
- [linear_equality_scale_types](#) : How to scale each linear equality constraint
- [linear_equality_scales](#) : Characteristic values to scale linear equalities
- [linear_equality_targets](#) : Define target values for the linear equality constraints
- [linear_inequality_constraint_matrix](#) : Define coefficients of the linear inequality constraints
- [linear_inequality_lower_bounds](#) : Define lower bounds for the linear inequality constraint
- [linear_inequality_scale_types](#) : How to scale each linear inequality constraint
- [linear_inequality_scales](#) : Characteristic values to scale linear inequalities
- [linear_inequality_upper_bounds](#) : Define upper bounds for the linear inequality constraint

6.5.5 variable_type

Description

Dakota variables can be grouped their type: [design](#), uncertain (further sub-divided into [aleatory](#) and [epistemic](#)), and [state](#). These distinctions are useful in situations where the user may want to explicitly control the subset of variables that is considered [active](#) for a certain Dakota method, overriding the default active variable type.

Related Topics

- [design_variables](#)
- [aleatory_uncertain_variables](#)
- [epistemic_uncertain_variables](#)
- [state_variables](#)

Related Keywords

6.5.6 design_variables

Description

Design variables are adjusted in the course of seeking an optimal design or an optimal set of deterministic calibration parameters.

Continuous design variables, which may assume any real value within specified bounds, are the most common design variable type in engineering applications. All but a handful of the optimization algorithms in Dakota support continuous design variables exclusively.

Related Topics

Related Keywords

- [continuous_design](#) : Design variable - continuous
- [discrete_design_range](#) : Design variable - discrete range-valued
- [discrete_design_set](#) : Design variable - discrete set-valued
- [integer](#) : Integer-valued discrete design variables
- [real](#) : Real-valued discrete design variables
- [string](#) : String-valued discrete design set variables

6.5.7 aleatory_uncertain_variables

Description

Aleatory uncertainty is also known as inherent variability, irreducible uncertainty, or randomness.

Aleatory uncertainty is typically represented by probability distributions and is specified to Dakota using the following parametric and histogram uncertain variable keywords.

Related Topics

Related Keywords

- [beta_uncertain](#) : Aleatory uncertain variable - beta
- [binomial_uncertain](#) : Aleatory uncertain discrete variable - binomial
- [exponential_uncertain](#) : Aleatory uncertain variable - exponential
- [frechet_uncertain](#) : Aleatory uncertain variable - Frechet
- [gamma_uncertain](#) : Aleatory uncertain variable - gamma
- [geometric_uncertain](#) : Aleatory uncertain discrete variable - geometric
- [gumbel_uncertain](#) : Aleatory uncertain variable - gumbel
- [histogram_bin_uncertain](#) : Aleatory uncertain variable - continuous histogram
- [histogram_point_uncertain](#) : Aleatory uncertain variable - discrete histogram
- [hypergeometric_uncertain](#) : Aleatory uncertain discrete variable - hypergeometric
- [lognormal_uncertain](#) : Aleatory uncertain variable - lognormal
- [loguniform_uncertain](#) : Aleatory uncertain variable - loguniform
- [negative_binomial_uncertain](#) : Aleatory uncertain discrete variable - negative binomial
- [normal_uncertain](#) : Aleatory uncertain variable - normal (Gaussian)
- [poisson_uncertain](#) : Aleatory uncertain discrete variable - Poisson
- [triangular_uncertain](#) : Aleatory uncertain variable - triangular
- [uniform_uncertain](#) : Aleatory uncertain variable - uniform
- [weibull_uncertain](#) : Aleatory uncertain variable - Weibull

6.5.8 epistemic_uncertain_variables

Description

Epistemic uncertainty is uncertainty due to lack of knowledge.

In Dakota, epistemic uncertainty can be characterized by interval- or set-valued variables (see relevant keywords below) that are propagated to calculate bounding intervals on simulation output using interval analysis methods. These epistemic variable types can optionally include basic probability assignments for use in Dempster-Shafer theory of evidence methods. Epistemic uncertainty can alternately be modeled with probability density functions, although results from UQ studies are then typically interpreted as possibilities or bounds, as opposed to a probability distribution of responses.

Through [nested models](#), Dakota can perform combined aleatory / epistemic analyses such as second-order probability or probability of frequency. For example, a variable can be assumed to have a lognormal distribution with specified variance, with its mean expressed as an epistemic uncertainty lying in an expert-specified interval.

Related Topics

Related Keywords

- [continuous_interval_uncertain](#) : Epistemic uncertain variable - values from one or more continuous intervals
- [discrete_interval_uncertain](#) : Epistemic uncertain variable - values from one or more discrete intervals
- [discrete_uncertain_set](#) : Epistemic uncertain variable - discrete set-valued
- [integer](#) : Discrete, epistemic uncertain variable - integers within a set
- [real](#) : Discrete, epistemic uncertain variable - real numbers within a set
- [string](#) : Discrete, epistemic uncertain variable - strings within a set

6.5.9 state_variables

Description

State variables provide a convenient mechanism for managing additional model parameterizations such as mesh density, simulation convergence tolerances, and time step controls. These are typically fixed parameters for a given Dakota run.

By default, only parameter studies and design of experiments methods will vary state variables. This can be overridden for other methods by specifying `active state` or `active all`.

When a state variable is held fixed, the specified `initial_state` is used as its sole value. If the state variable is defined only by its bounds, then the `initial_state` will be inferred from the variable bounds or valid set values.

If a method iterates on a state variable, the variable is treated as a design variable with the given bounds, or as a uniform uncertain variable with the given bounds.

Related Topics

Related Keywords

- [continuous_state](#) : State variable - continuous
- [discrete_state_range](#) : State variables - discrete range-valued
- [discrete_state_set](#) : State variable - discrete set-valued
- [integer](#) : Discrete state variables, each defined by a set of permissible integers
- [real](#) : Discrete state variables, each defined by a set of permissible real numbers
- [string](#) : String-valued discrete state set variables

6.6 responses

Description

Keywords and concepts relating to the `responses` block

Related Topics

- [response_types](#)
- [nonlinear_constraints](#)

Related Keywords

6.6.1 `response_types`

Description

The specification must be one of three types:

1. objective and constraint functions
2. calibration (least squares) terms and constraint functions
3. a generic response functions specification.

These correspond to (a) optimization, (b) deterministic (least squares) or stochastic (Bayesian) inversion, and (c) general-purpose analyzer methods such as parameter studies, DACE, and UQ methods, respectively. Refer to [responses](#) for additional details and examples.

Related Topics

Related Keywords

6.6.2 `nonlinear_constraints`

Description

Nonlinear constraints are supported by many of Dakota's optimizers. For a discussion of the difference between nonlinear and linear constraints, see the [linear_constraints](#) topic page. The Constraint Considerations section of the [optimization_and_calibration](#) page may also be of interest.

Related Topics

Related Keywords

- [nonlinear_equality_constraints](#) : Group to specify nonlinear equality constraints
- [nonlinear_inequality_constraints](#) : Group to specify nonlinear inequality constraints
- [nonlinear_equality_constraints](#) : Group to specify nonlinear equality constraints
- [nonlinear_inequality_constraints](#) : Group to specify nonlinear inequality constraints

6.7 `interface`

Description

Keywords and Concepts relating to the `interface` block, which is used to connect Dakota to external analysis codes (simulations, etc.)

Related Topics

- [simulation_file_management](#)
- [workflow_management](#)
- [advanced_simulation_interfaces](#)

Related Keywords**6.7.1 simulation_file_management****Description**

empty

Related Topics**Related Keywords****6.7.2 workflow_management****Description**

empty

Related Topics**Related Keywords****6.7.3 advanced_simulation_interfaces****Description**

empty

Related Topics

- [simulation_failure](#)
- [concurrency_and_parallelism](#)

Related Keywords**6.7.4 simulation_failure****Description**

empty

Related Topics**Related Keywords****6.7.5 concurrency_and_parallelism****Description**

empty

Related Topics

Related Keywords

- [processors_per_analysis](#) : Specify the number of processors per analysis when Dakota is run in parallel
- [analysis_scheduling](#) : Specify the scheduling of concurrent analyses when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel analysis scheduling
- [peer](#) : Specify a peer partition for parallel analysis scheduling
- [analysis_servers](#) : Specify the number of analysis servers when Dakota is run in parallel
- [asynchronous](#) : Specify local evaluation or analysis concurrency
- [analysis_concurrency](#) : Limit the number of analysis drivers within an evaluation that Dakota will schedule
- [evaluation_concurrency](#) : Determine how many concurrent evaluations Dakota will schedule
- [local_evaluation_scheduling](#) : Control how local asynchronous jobs are scheduled
- [batch](#) : Perform evaluations in batches
- [size](#) : Limit the number of evaluations in a batch
- [master](#) : Specify a dedicated master partition for parallel evaluation scheduling
- [peer](#) : Specify a peer partition for parallel evaluation scheduling
- [dynamic](#) : Specify dynamic scheduling in a peer partition when Dakota is run in parallel.
- [static](#) : Specify static scheduling in a peer partition when Dakota is run in parallel.
- [evaluation_servers](#) : Specify the number of evaluation servers when Dakota is run in parallel
- [processors_per_evaluation](#) : Specify the number of processors per evaluation server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel

- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel

6.8 methods

Description

Keywords and Concepts relating to the `method` block, including discussion of the different methods and algorithms available in Dakota

Related Topics

- [parameter_studies](#)
- [sensitivity_analysis_and_design_of_experiments](#)
- [uncertainty_quantification](#)
- [optimization_and_calibration](#)

Related Keywords

6.8.1 parameter_studies

Description

Parameter studies employ deterministic designs to explore the effect of parametric changes within simulation models, yielding one form of sensitivity analysis. They can help assess simulation characteristics such as smoothness, multi-modality, robustness, and nonlinearity, which affect the choice of algorithms and controls in follow-on optimization and UQ studies.

Dakota's parameter study methods compute response data sets at a selection of points in the parameter space. These points may be specified as a vector, a list, a set of centered vectors, or a multi-dimensional grid. Capability overviews and examples of the different types of parameter studies are provided in the Users Manual [5].

With the exception of output verbosity (a setting of `silent` will suppress some parameter study diagnostic output), Dakota's parameter study methods do not make use of the method independent controls. Therefore, the parameter study documentation which follows is limited to the method dependent controls for the vector, list, centered, and multidimensional parameter study methods.

Related Topics

Related Keywords

- [centered_parameter_study](#) : Samples variables along points moving out from a center point
- [list_parameter_study](#) : Samples variables as a specified values
- [multidim_parameter_study](#) : Samples variables on full factorial grid of study points
- [partitions](#) : Samples variables on full factorial grid of study points
- [vector_parameter_study](#) : Samples variables along a user-defined vector

6.8.2 sensitivity_analysis_and_design_of_experiments

Description

empty

Related Topics

- [design_and_analysis_of_computer_experiments](#)
- [sampling](#)

Related Keywords

6.8.3 design_and_analysis_of_computer_experiments

Description

Design and Analysis of Computer Experiments (DACE) methods compute response data sets at a selection of points in the parameter space. Three libraries are provided for performing these studies: DDACE, FSUDace, and PSUADE. The design of experiments methods do not currently make use of any of the method independent controls.

Related Topics

Related Keywords

- [dace](#) : Design and Analysis of Computer Experiments
- [fsu_cvt](#) : Design of Computer Experiments - Centroidal Voronoi Tessellation
- [fsu_quasi_mc](#) : Design of Computer Experiments - Quasi-Monte Carlo sampling
- [hammersley](#) : Use Hammersley sequences
- [psuade_moat](#) : Morris One-at-a-Time

6.8.4 sampling

Description

Sampling techniques are selected using the `sampling` method selection. This method generates sets of samples according to the probability distributions of the uncertain variables and maps them into corresponding sets of response functions, where the number of samples is specified by the `samples` integer specification. Means, standard deviations, coefficients of variation (COVs), and 95% confidence intervals are computed for the response functions. Probabilities and reliabilities may be computed for `response_levels` specifications, and response levels may be computed for either `probability_levels` or `reliability_levels` specifications (refer to the Method Commands chapter in the Dakota Reference Manual [4] for additional information).

Currently, traditional Monte Carlo (MC) and Latin hypercube sampling (LHS) are supported by Dakota and are chosen by specifying `sample_type` as `random` or `lhs`. In Monte Carlo sampling, the samples are selected randomly according to the user-specified probability distributions. Latin hypercube sampling is a stratified sampling technique for which the range of each uncertain variable is divided into N_s segments of equal probability, where N_s is the number of samples requested. The relative lengths of the segments are determined by the nature of the specified probability distribution (e.g., uniform has segments of equal width, normal has small segments near the mean and larger segments in the tails). For each of the uncertain variables, a sample is selected randomly from each of these equal probability segments. These N_s values for each of the individual parameters are then combined in a shuffling operation to create a set of N_s parameter vectors with a specified correlation structure. A feature of the resulting sample set is that *every row and column in the hypercube of partitions has exactly one sample*. Since the total number of samples is exactly equal to the number of partitions used for each uncertain variable, an arbitrary number of desired samples is easily accommodated (as compared to less flexible approaches in which the total number of samples is a product or exponential function of the number of intervals for each variable, i.e., many classical design of experiments methods).

Advantages of sampling-based methods include their relatively simple implementation and their independence from the scientific disciplines involved in the analysis. The main drawback of these techniques is the large number of function evaluations needed to generate converged statistics, which can render such an analysis computationally very expensive, if not intractable, for real-world engineering applications. LHS techniques, in general, require fewer samples than traditional Monte Carlo for the same accuracy in statistics, but they still can be prohibitively expensive. For further information on the method and its relationship to other sampling techniques, one is referred to the works by McKay, et al. [61], Iman and Shortencarier [54], and Helton and Davis [47]. Note that under certain separability conditions associated with the function to be sampled, Latin hypercube sampling provides a more accurate estimate of the mean value than does random sampling. That is, given an equal number of samples, the LHS estimate of the mean will have less variance than the mean value obtained through random sampling.

Related Topics

Related Keywords

- [importance_sampling](#) : Importance sampling
- [sampling](#) : Randomly samples variables according to their distributions

6.8.5 uncertainty_quantification

Description

Dakota provides a variety of methods for propagating both aleatory and epistemic uncertainty.

At a high level, uncertainty quantification (UQ) or nondeterministic analysis is the process of characterizing input uncertainties, forward propagating these uncertainties through a computational model, and performing statistical or interval assessments on the resulting responses. This process determines the effect of uncertainties and assumptions on model outputs or results. In Dakota, uncertainty quantification methods specifically focus on the forward propagation part of the process, where probabilistic or interval information on parametric inputs are mapped through

the computational model to assess statistics or intervals on outputs. For an overview of these approaches for engineering applications, consult [43].

UQ is related to sensitivity analysis in that the common goal is to gain an understanding of how variations in the parameters affect the response functions of the engineering design problem. However, for UQ, some or all of the components of the parameter vector, are considered to be uncertain as specified by particular probability distributions (e.g., normal, exponential, extreme value), or other uncertainty structures. By assigning specific distributional structure to the inputs, distributional structure for the outputs (i.e., response statistics) can be inferred. This migrates from an analysis that is more *{qualitative}* in nature, in the case of sensitivity analysis, to an analysis that is more rigorously *{quantitative}*.

UQ methods are often distinguished by their ability to propagate aleatory or epistemic input uncertainty characterizations, where aleatory uncertainties are irreducible variabilities inherent in nature and epistemic uncertainties are reducible uncertainties resulting from a lack of knowledge. Since sufficient data is generally available for aleatory uncertainties, probabilistic methods are commonly used for computing response distribution statistics based on input probability distribution specifications. Conversely, for epistemic uncertainties, any use of probability distributions is based on subjective knowledge rather than objective data, and we may alternatively explore nonprobabilistic methods based on interval specifications.

Dakota contains capabilities for performing nondeterministic analysis with both types of input uncertainty. These UQ methods have been developed by Sandia Labs, in conjunction with collaborators in academia [31], [32], [21], [82].

The aleatory UQ methods in Dakota include various sampling-based approaches (e.g., Monte Carlo and Latin Hypercube sampling), local and global reliability methods, and stochastic expansion (polynomial chaos expansions and stochastic collocation) approaches. The epistemic UQ methods include local and global interval analysis and Dempster-Shafer evidence theory. These are summarized below and then described in more depth in subsequent sections of this chapter. Dakota additionally supports mixed aleatory/epistemic UQ via interval-valued probability, second-order probability, and Dempster-Shafer theory of evidence. These involve advanced model recursions and are described in Section.

Dakota contains capabilities for performing nondeterministic analysis with both types of input uncertainty. These UQ methods have been developed by Sandia Labs, in conjunction with collaborators in academia [31], [32], [21], [82].

The aleatory UQ methods in Dakota include various sampling-based approaches (e.g., Monte Carlo and Latin Hypercube sampling), local and global reliability methods, and stochastic expansion (polynomial chaos expansions and stochastic collocation) approaches. The epistemic UQ methods include local and global interval analysis and Dempster-Shafer evidence theory. These are summarized below and then described in more depth in subsequent sections of this chapter. Dakota additionally supports mixed aleatory/epistemic UQ via interval-valued probability, second-order probability, and Dempster-Shafer theory of evidence. These involve advanced model recursions and are described in Section.

The choice of uncertainty quantification method depends on how the input uncertainty is characterized, the computational budget, and the desired output accuracy. The recommendations for UQ methods are summarized in Table and are discussed in the remainder of the section.

TODO: Put table in Doxygen if still needed

Related Topics

- [aleatory_uncertainty_quantification_methods](#)
- [epistemic_uncertainty_quantification_methods](#)
- [variable_support](#)

Related Keywords

- [adaptive_sampling](#) : (Experimental) Adaptively refine a Gaussian process surrogate
- [global_interval_est](#) : Interval analysis using global optimization methods

- [global_reliability](#) : Global reliability methods
- [gpais](#) : Gaussian Process Adaptive Importance Sampling
- [importance_sampling](#) : Importance sampling
- [local_interval_est](#) : Interval analysis using local optimization
- [local_reliability](#) : Local reliability method
- [mpp_search](#) : Specify which MPP search option to use
- [pof_darts](#) : Probability-of-Failure (POF) darts is a novel method for estimating the probability of failure based on random sphere-packing.
- [rkd_darts](#) : Recursive k-d (RKD) Darts: Recursive Hyperplane Sampling for Numerical Integration of High-Dimensional Functions.
- [sampling](#) : Randomly samples variables according to their distributions

6.8.6 aleatory_uncertainty_quantification_methods

Description

Aleatory uncertainty is also known as inherent variability, irreducible uncertainty, or randomness.

Aleatory uncertainty is typically characterized using probability theory.

Related Topics

- [sampling](#)
- [reliability_methods](#)
- [stochastic_expansion_methods](#)

Related Keywords

- [importance_sampling](#) : Importance sampling

6.8.7 sampling

Description

Sampling techniques are selected using the `sampling` method selection. This method generates sets of samples according to the probability distributions of the uncertain variables and maps them into corresponding sets of response functions, where the number of samples is specified by the `samples` integer specification. Means, standard deviations, coefficients of variation (COVs), and 95% confidence intervals are computed for the response functions. Probabilities and reliabilities may be computed for `response_levels` specifications, and response levels may be computed for either `probability_levels` or `reliability_levels` specifications (refer to the Method Commands chapter in the Dakota Reference Manual [4] for additional information).

Currently, traditional Monte Carlo (MC) and Latin hypercube sampling (LHS) are supported by Dakota and are chosen by specifying `sample_type` as `random` or `lhs`. In Monte Carlo sampling, the samples are selected randomly according to the user-specified probability distributions. Latin hypercube sampling is a stratified sampling technique for which the range of each uncertain variable is divided into N_s segments of equal probability, where N_s is the number of samples requested. The relative lengths of the segments are determined by the nature of the specified probability distribution (e.g., uniform has segments of equal width, normal has small segments near the mean and larger segments in the tails). For each of the uncertain variables, a sample is selected randomly from

each of these equal probability segments. These N_s values for each of the individual parameters are then combined in a shuffling operation to create a set of N_s parameter vectors with a specified correlation structure. A feature of the resulting sample set is that *every row and column in the hypercube of partitions has exactly one sample*. Since the total number of samples is exactly equal to the number of partitions used for each uncertain variable, an arbitrary number of desired samples is easily accommodated (as compared to less flexible approaches in which the total number of samples is a product or exponential function of the number of intervals for each variable, i.e., many classical design of experiments methods).

Advantages of sampling-based methods include their relatively simple implementation and their independence from the scientific disciplines involved in the analysis. The main drawback of these techniques is the large number of function evaluations needed to generate converged statistics, which can render such an analysis computationally very expensive, if not intractable, for real-world engineering applications. LHS techniques, in general, require fewer samples than traditional Monte Carlo for the same accuracy in statistics, but they still can be prohibitively expensive. For further information on the method and its relationship to other sampling techniques, one is referred to the works by McKay, et al. [61], Iman and Shortencarier [54], and Helton and Davis [47]. Note that under certain separability conditions associated with the function to be sampled, Latin hypercube sampling provides a more accurate estimate of the mean value than does random sampling. That is, given an equal number of samples, the LHS estimate of the mean will have less variance than the mean value obtained through random sampling.

Related Topics

Related Keywords

- [importance_sampling](#) : Importance sampling
- [sampling](#) : Randomly samples variables according to their distributions

6.8.8 reliability_methods

Description

Reliability methods provide an alternative approach to uncertainty quantification which can be less computationally demanding than sampling techniques. Reliability methods for uncertainty quantification are based on probabilistic approaches that compute approximate response function distribution statistics based on specified uncertain variable distributions. These response statistics include response mean, response standard deviation, and cumulative or complementary cumulative distribution functions (CDF/CCDF). These methods are often more efficient at computing statistics in the tails of the response distributions (events with low probability) than sampling based approaches since the number of samples required to resolve a low probability can be prohibitive.

The methods all answer the fundamental question: “Given a set of uncertain input variables, \mathbf{X} , and a scalar response function, g , what is the probability that the response function is below or above a certain level, \bar{z} ?” The former can be written as $P[g(\mathbf{X}) \leq \bar{z}] = F_g(\bar{z})$ where $F_g(\bar{z})$ is the cumulative distribution function (CDF) of the uncertain response $g(\mathbf{X})$ over a set of response levels. The latter can be written as $P[g(\mathbf{X}) > \bar{z}]$ and defines the complementary cumulative distribution function (CCDF).

This probability calculation involves a multi-dimensional integral over an irregularly shaped domain of interest, \mathbf{D} , where $g(\mathbf{X}) < z$ as displayed in Figure [figUQ05](#) for the case of two variables. The reliability methods all involve the transformation of the user-specified uncertain variables, \mathbf{X} , with probability density function, $p(x_1, x_2)$, which can be non-normal and correlated, to a space of independent Gaussian random variables, \mathbf{u} , possessing a mean value of zero and unit variance (i.e., standard normal variables). The region of interest, \mathbf{D} , is also mapped to the transformed space to yield, \mathbf{D}_u , where $g(\mathbf{U}) < z$ as shown in Figure [figUQ06](#). The Nataf transformation [17], which is identical to the Rosenblatt transformation [75] in the case of independent random variables, is used in Dakota to accomplish this mapping. This transformation is performed to make the probability calculation more tractable. In the transformed space, probability contours are circular in nature as shown in Figure [figUQ06](#) unlike in the original uncertain variable space, Figure [figUQ05](#). Also, the multi-dimensional integrals can be approximated by simple functions of a single parameter, β , called the reliability index. β is the minimum Euclidean distance from the origin in the transformed space to the response surface. This point is also known as the most probable point (MPP) of failure. Note, however, the methodology is equally applicable for generic functions, not simply those corresponding

to failure criteria; this nomenclature is due to the origin of these methods within the disciplines of structural safety and reliability. Note that there are local and global reliability methods. The majority of the methods available are local, meaning that a local optimization formulation is used to locate one MPP. In contrast, global methods can find multiple MPPs if they exist.

Related Topics

Related Keywords

- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [global_reliability](#) : Global reliability methods
- [u_gaussian_process](#) : Create GP surrogate in u-space
- [x_gaussian_process](#) : Create GP surrogate in x-space
- [local_reliability](#) : Local reliability method
- [mpp_search](#) : Specify which MPP search option to use
- [integration](#) : Integration approach
- [first_order](#) : First-order integration scheme
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [second_order](#) : Second-order integration scheme
- [no_approx](#) : Perform MPP search on original response functions (use no approximation)
- [u_taylor_mean](#) : Form Taylor series approximation in "u-space" at variable means
- [u_taylor_mpp](#) : U-space Taylor series approximation with iterative updates
- [u_two_point](#) : Predict MPP using Two-point Adaptive Nonlinear Approximation in "u-space"
- [x_taylor_mean](#) : Form Taylor series approximation in "x-space" at variable means
- [x_taylor_mpp](#) : X-space Taylor series approximation with iterative updates
- [x_two_point](#) : Predict MPP using Two-point Adaptive Nonlinear Approximation in "x-space"
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling
- [probability_refinement](#) : Allow refinement of probability and generalized reliability results using importance sampling

6.8.9 stochastic_expansion_methods

Description

The development of these techniques mirrors that of deterministic finite element analysis utilizing the notions of projection, orthogonality, and weak convergence [31], [32]. Rather than estimating point probabilities, they form an approximation to the functional relationship between response functions and their random inputs, which provides a more complete uncertainty representation for use in multi-code simulations. Expansion methods include polynomial chaos expansions (PCE), which employ multivariate orthogonal polynomials that are tailored to representing particular input probability distributions, and stochastic collocation (SC), which employs multivariate interpolation polynomials. For PCE, expansion coefficients may be evaluated using a spectral projection approach (based on sampling, tensor-product quadrature, Smolyak sparse grid, or cubature methods for numerical integration) or a regression approach (least squares or compressive sensing). For SC, interpolants are formed over tensor-product or sparse grids and may be local or global, value-based or gradient-enhanced, and nodal or hierarchical. In global value-based cases (Lagrange polynomials), the barycentric formulation is used [9], [57], [50] to improve numerical efficiency and stability. Both sets of methods provide analytic response moments and variance-based metrics; however, CDF/CCDF probabilities are evaluated numerically by sampling on the expansion.

Related Topics

Related Keywords

6.8.10 epistemic_uncertainty_quantification_methods

Description

Epistemic uncertainty is uncertainty due to lack of knowledge.

In Dakota, epistemic uncertainty analysis is performed using interval analysis or Dempster-Shafer theory of evidence.

Note that epistemic uncertainty can also be modeled probabilistically. It would be more accurate to call this class of method, non-probabilistic uncertainty quantification, but the name persists for historical reasons.

Related Topics

- [interval_estimation](#)
- [evidence_theory](#)

Related Keywords

- [global_evidence](#) : Evidence theory with evidence measures computed with global optimization methods
- [global_interval_est](#) : Interval analysis using global optimization methods
- [local_evidence](#) : Evidence theory with evidence measures computed with local optimization methods
- [local_interval_est](#) : Interval analysis using local optimization

6.8.11 interval_estimation

Description

In interval analysis, one assumes that nothing is known about an epistemic uncertain variable except that its value lies somewhere within an interval. In this situation, it is NOT assumed that the value has a uniform probability of

occurring within the interval. Instead, the interpretation is that any value within the interval is a possible value or a potential realization of that variable. In interval analysis, the uncertainty quantification problem is one of determining the resulting bounds on the output (defining the output interval) given interval bounds on the inputs. Again, any output response that falls within the output interval is a possible output with no frequency information assigned to it.

We have the capability to perform interval analysis using either `global_interval_est` or `local_interval_est`. In the global approach, one uses either a global optimization method or a sampling method to assess the bounds. `global_interval_est` allows the user to specify either `lhs`, which performs Latin Hypercube Sampling and takes the minimum and maximum of the samples as the bounds (no optimization is performed) or `ego`. In the case of `ego`, the efficient global optimization method is used to calculate bounds. The `ego` method is described in Section . If the problem is amenable to local optimization methods (e.g. can provide derivatives or use finite difference method to calculate derivatives), then one can use local methods to calculate these bounds. `local_interval_est` allows the user to specify either `sqp` which is sequential quadratic programming, or `nip` which is a nonlinear interior point method.

Note that when performing interval analysis, it is necessary to define interval uncertain variables as described in Section . For interval analysis, one must define only one interval per input variable, in contrast with Dempster-Shafer evidence theory, where an input can have several possible intervals. Interval analysis can be considered a special case of Dempster-Shafer evidence theory where each input is defined by one input interval with a basic probability assignment of one. In Dakota, however, the methods are separate and semantic differences exist in the output presentation. If you are performing a pure interval analysis, we recommend using either `global_interval_est` or `local_interval_est` instead of `global_evidence` or `local_evidence`, for reasons of simplicity. An example of interval estimation is found in the `dakota/share/dakota/examples/users/cantilever_uq_global_interval.in`, and also in Section .

Note that we have kept separate implementations of interval analysis and Dempster-Shafer evidence theory because our users often want to couple interval analysis on an outer loop” with an aleatory, probabilistic analysis on an inner loop” for nested, second-order probability calculations. See Section for additional details on these nested approaches. These interval methods can also be used as the outer loop within an interval-valued probability analysis for propagating mixed aleatory and epistemic uncertainty – refer to Section for additional details.

Interval analysis is often used to model epistemic uncertainty. In interval analysis, the uncertainty quantification problem is one of determining the resulting bounds on the output (defining the output interval) given interval bounds on the inputs.

We can do interval analysis using either `%global_interval_est` or `local_interval_est`. In the global approach, one uses either a global optimization method or a sampling method to assess the bounds, whereas the local method uses gradient information in a derivative-based optimization approach.

An example of interval estimation is shown in Figure , with example results in Figure . This example is a demonstration of calculating interval bounds for three outputs of the cantilever beam problem. The cantilever beam problem is described in detail in Section . Given input intervals of [1,10] on beam width and beam thickness, we can see that the interval estimate of beam weight is approximately [1,100].

:examples:interval_out

```
-----
Min and Max estimated values for each response function:
weight:  Min = 1.0000169352e+00  Max = 9.9999491948e+01
stress:  Min = -9.7749994284e-01  Max = 2.1499428450e+01
displ:   Min = -9.9315672724e-01  Max = 6.7429714485e+01
-----
```

Related Topics

Related Keywords

- [global_interval_est](#) : Interval analysis using global optimization methods
- [local_interval_est](#) : Interval analysis using local optimization

6.8.12 evidence_theory

Description

This section discusses Dempster-Shafer evidence theory. In this approach, one does not assign a probability distribution to each uncertain input variable. Rather, one divides each uncertain input variable into one or more intervals. The input parameters are only known to occur within intervals: nothing more is assumed.

Each interval is defined by its upper and lower bounds, and a Basic Probability Assignment (BPA) associated with that interval. The BPA represents a probability of that uncertain variable being located within that interval.

The intervals and BPAs are used to construct uncertainty measures on the outputs called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. For more information about the Dempster-Shafer theory of evidence, see [69] and [48].

Similar to the interval approaches, one may use global or local methods to determine plausibility and belief measures for the outputs.

Usage Notes

Note that to calculate the plausibility and belief cumulative distribution functions, one has to look at all combinations of intervals for the uncertain variables. Within each interval cell combination, the minimum and maximum value of the objective function determine the belief and plausibility, respectively. In terms of implementation, global methods use LHS sampling or global optimization to calculate the minimum and maximum values of the objective function within each interval cell, while local methods use gradient-based optimization methods to calculate these minima and maxima.

Finally, note that many non-deterministic keywords apply to the evidence methods, but one needs to be careful about the interpretation and translate probabilistic measures to epistemic ones. For example, if the user specifies distribution of type complementary, a complementary plausibility and belief function will be generated for the evidence methods (as opposed to a complementary distribution function in the `sampling` case). If the user specifies a set of responses levels, both the belief and plausibility will be calculated for each response level. Likewise, if the user specifies a probability level, the probability level will be interpreted both as a belief and plausibility, and response levels corresponding to the belief and plausibility levels will be calculated. Finally, if generalized reliability levels are specified, either as inputs (`gen_reliability_levels`) or outputs (`response_levels` with `compute_gen_reliabilities`), then these are directly converted to/from probability levels and the same probability-based mappings described above are performed.

Related Topics

Related Keywords

- [global_evidence](#) : Evidence theory with evidence measures computed with global optimization methods
- [local_evidence](#) : Evidence theory with evidence measures computed with local optimization methods

6.8.13 variable_support

Description

Different nondeterministic methods have differing support for uncertain variable distributions. Tables 5.37, 5.38, and 5.39 summarize the uncertain variables that are available for use by the different methods, where a "-" indicates that the distribution is not supported by the method, a "U" means the uncertain input variables of this type must be uncorrelated, a "C" denotes that correlations are supported involving uncertain input variables of this type, and an "A" means the appropriate variables must be specified as active in the variables specification block. For example, if one wants to support sampling or a stochastic expansion method over both continuous uncertain and continuous state variables, the specification `active all` must be listed in the variables specification block. Additional notes include:

- we have four variants for stochastic expansions (SE), listed as Wiener, Askey, Extended, and Piecewise which draw from different sets of basis polynomials. The term stochastic expansion indicates polynomial chaos and stochastic collocation collectively, although the Piecewise option is only currently supported for stochastic collocation. Refer to [polynomial_chaos](#) and [stoch_collocation](#) for additional information on these three options.
- methods supporting the epistemic interval distributions have differing approaches: `sampling` and the `lhs` option of `global_interval_est` model the interval basic probability assignments (BPAs) as continuous histogram bin distributions for purposes of generating samples; `local_interval_est` and the `ego` option of `global_interval_est` ignore the BPA details and models these variables as simple bounded regions defined by the cell extremes; and `local_evidence` and `global_evidence` model the interval specifications as true BPAs.

Distribution Type	Sampling	Local Reliability	Global Reliability	Wiener SE	Askey SE	Extended SE	Piecewise SE	Local Interval	Global Interval	Local Evidence	Global Evidence
Normal	C	C	C	C	C	C	-	-	-	-	-
Bounded Normal	C	U	U	U	U	U	U	-	-	-	-
Log-normal	C	C	C	C	C	U	-	-	-	-	-
Bounded Log-normal	C	U	U	U	U	U	U	-	-	-	-
Uniform	C	C	C	C	U	U	U	-	-	-	-
Loguniform	C	U	U	U	U	U	U	-	-	-	-
Triangular	C	U	U	U	U	U	U	-	-	-	-
Exponential	C	C	C	C	U	U	-	-	-	-	-
Beta	C	U	U	U	U	U	U	-	-	-	-
Gamma	C	C	C	C	U	U	-	-	-	-	-
Gumbel	C	C	C	C	C	U	-	-	-	-	-
Frechet	C	C	C	C	C	U	-	-	-	-	-
Weibull	C	C	C	C	C	U	-	-	-	-	-
Continuous Histogram Bin	C	U	U	U	U	U	U	-	-	-	-

Table 6.1: Summary of Distribution Types supported by Nondeterministic Methods, Part I (Continuous Aleatory Types)

Distribution Type	Sampling	Local Reliability	Global Reliability	Wiener SE	Askey SE	Extended SE	Piecewise SE	Local Interval	Global Interval	Local Evidence	Global Evidence
Poisson	C	-	-	-	-	-	-	-	-	-	-
Binomial	C	-	-	-	-	-	-	-	-	-	-
Negative Binomial	C	-	-	-	-	-	-	-	-	-	-
Geometric	C	-	-	-	-	-	-	-	-	-	-
Hypergeometric	C	-	-	-	-	-	-	-	-	-	-
Discrete Histogram Point	C	-	-	-	-	-	-	-	-	-	-

Table 6.2: Summary of Distribution Types supported by Nondeterministic Methods, Part II (Discrete Aleatory Types)

Related Topics

Related Keywords

6.8.14 optimization_and_calibration

Description

Optimization algorithms work to minimize (or maximize) an objective function, typically calculated by the user simulation code, subject to constraints on design variables and responses. Available approaches in Dakota include well-tested, proven gradient-based, derivative-free local, and global methods for use in science and engineering design applications. Dakota also offers more advanced algorithms, e.g., to manage multi-objective optimization or perform surrogate-based minimization. This chapter summarizes optimization problem formulation, standard algorithms available in Dakota (mostly through included third-party libraries, see Section 6.5 of [5]), some advanced capabilities, and offers usage guidelines.

6.8.14.1 Optimization Formulations

This section provides a basic introduction to the mathematical formulation of optimization, problems. The primary goal of this section is to introduce terms relating to these topics, and is not intended to be a description of theory or numerical algorithms. For further details, consult [7], [34], [42], [67], and [87].

A general optimization problem is formulated as follows:

Distribution Type	Sampling	Local Reliability	Global Reliability	Wiener SE	Askey SE	Extended SE	Piecewise SE	Local Interval	Global Interval	Local Evidence	Global Evidence
Interval	U	-	U,A	U,A	U,A	U,A	U,A	U	U	U	U
Continuous Design	U,A	-	U,A	U,A	U,A	U,A	U,A	-	-	-	-
Discrete Design Range, Int Set, Real Set	U,A	-	-	-	-	-	-	-	-	-	-
Continuous State	U,A	-	U,A	U,A	U,A	U,A	U,A	-	-	-	-
Discrete State Range, Int Set, Real Set	U,A	-	-	-	-	-	-	-	-	-	-

Table 6.3: Summary of Distribution Types supported by Nondeterministic Methods, Part III (Epistemic, Design, and State Types)

$$\begin{aligned}
 &\text{minimize:} && f(\mathbf{x}) \\
 &&& \mathbf{x} \in \mathfrak{X}^n \\
 &\text{subject to:} && \mathbf{g}_L \leq \mathbf{g}(\mathbf{x}) \leq \mathbf{g}_U \\
 &&& \mathbf{h}(\mathbf{x}) = \mathbf{h}_t \\
 &&& \mathbf{a}_L \leq \mathbf{A}_i \mathbf{x} \leq \mathbf{a}_U \\
 &&& \mathbf{A}_e \mathbf{x} = \mathbf{a}_t \\
 &&& \mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U
 \end{aligned} \tag{6.1}$$

where vector and matrix terms are marked in bold typeface. In this formulation, $\mathbf{x} = [x_1, x_2, \dots, x_n]$ is an n -dimensional vector of real-valued *design variables* or *design parameters*. The n -dimensional vectors, \mathbf{x}_L and \mathbf{x}_U , are the lower and upper bounds, respectively, on the design parameters. These bounds define the allowable values for the elements of \mathbf{x} , and the set of all allowable values is termed the *design space* or the *parameter space*. A *design point* or a *sample point* is a particular set of values within the parameter space.

The optimization goal is to minimize the *objective function*, $f(\mathbf{x})$, while satisfying the constraints. Constraints can be categorized as either linear or nonlinear and as either inequality or equality. The *nonlinear inequality constraints*, $\mathbf{g}(\mathbf{x})$, are “2-sided,” in that they have both lower and upper bounds, \mathbf{g}_L and \mathbf{g}_U , respectively. The *nonlinear equality constraints*, $\mathbf{h}(\mathbf{x})$, have target values specified by \mathbf{h}_t . The linear inequality constraints create a linear system $\mathbf{A}_i \mathbf{x}$, where \mathbf{A}_i is the coefficient matrix for the linear system. These constraints are also 2-sided as they have lower and upper bounds, \mathbf{a}_L and \mathbf{a}_U , respectively. The linear equality constraints create a linear system $\mathbf{A}_e \mathbf{x}$, where \mathbf{A}_e is the

coefficient matrix for the linear system and \mathbf{a}_i are the target values. The constraints partition the parameter space into feasible and infeasible regions. A design point is said to be *feasible* if and only if it satisfies all of the constraints. Correspondingly, a design point is said to be *infeasible* if it violates one or more of the constraints.

Many different methods exist to solve the optimization problem given in Section 6.1 of [5], all of which iterate on \mathbf{x} in some manner. That is, an initial value for each parameter in \mathbf{x} is chosen, the *response quantities*, $f(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$, $\mathbf{h}(\mathbf{x})$, are computed, often by running a simulation, and some algorithm is applied to generate a new \mathbf{x} that will either reduce the objective function, reduce the amount of infeasibility, or both. To facilitate a general presentation of these methods, three criteria will be used in the following discussion to differentiate them: optimization problem type, search goal, and search method.

The **optimization problem type** can be characterized both by the types of constraints present in the problem and by the linearity or nonlinearity of the objective and constraint functions. For constraint categorization, a hierarchy of complexity exists for optimization algorithms, ranging from simple bound constraints, through linear constraints, to full nonlinear constraints. By the nature of this increasing complexity, optimization problem categorizations are inclusive of all constraint types up to a particular level of complexity. That is, an *unconstrained problem* has no constraints, a *bound-constrained problem* has only lower and upper bounds on the design parameters, a *linearly-constrained problem* has both linear and bound constraints, and a *nonlinearly-constrained problem* may contain the full range of nonlinear, linear, and bound constraints. If all of the linear and nonlinear constraints are equality constraints, then this is referred to as an *equality-constrained problem*, and if all of the linear and nonlinear constraints are inequality constraints, then this is referred to as an *inequality-constrained problem*. Further categorizations can be made based on the linearity of the objective and constraint functions. A problem where the objective function and all constraints are linear is called a *linear programming (LP) problem*. These types of problems commonly arise in scheduling, logistics, and resource allocation applications. Likewise, a problem where at least some of the objective and constraint functions are nonlinear is called a *nonlinear programming (NLP) problem*. These NLP problems predominate in engineering applications and are the primary focus of Dakota.

The **search goal** refers to the ultimate objective of the optimization algorithm, i.e., either global or local optimization. In *global optimization*, the goal is to find the design point that gives the lowest feasible objective function value over the entire parameter space. In contrast, in *local optimization*, the goal is to find a design point that is lowest relative to a “nearby” region of the parameter space. In almost all cases, global optimization will be more computationally expensive than local optimization. Thus, the user must choose an optimization algorithm with an appropriate search scope that best fits the problem goals and the computational budget.

The **search method** refers to the approach taken in the optimization algorithm to locate a new design point that has a lower objective function or is more feasible than the current design point. The search method can be classified as either *gradient-based* or *nongradient-based*. In a gradient-based algorithm, gradients of the response functions are computed to find the direction of improvement. Gradient-based optimization is the search method that underlies many efficient local optimization methods. However, a drawback to this approach is that gradients can be computationally expensive, inaccurate, or even nonexistent. In such situations, nongradient-based search methods may be useful. There are numerous approaches to nongradient-based optimization. Some of the more well known of these include pattern search methods (nongradient-based local techniques) and genetic algorithms (nongradient-based global techniques).

Because of the computational cost of running simulation models, surrogate-based optimization (SBO) methods are often used to reduce the number of actual simulation runs. In SBO, a surrogate or approximate model is constructed based on a limited number of simulation runs. The optimization is then performed on the surrogate model. Dakota has an extensive framework for managing a variety of local, multipoint, global, and hierarchical surrogates for use in optimization. Finally, sometimes there are multiple objectives that one may want to optimize simultaneously instead of a single scalar objective. In this case, one may employ multi-objective methods that are described in Section 6.3.1 of [5].

This overview of optimization approaches underscores that no single optimization method or algorithm works best for all types of optimization problems. Section 6.4 of [5] offers guidelines for choosing a Dakota optimization algorithm best matched to your specific optimization problem.

6.8.14.1.1 Constraint Considerations

Dakota’s input commands permit the user to specify two-sided nonlinear inequality constraints of the form $g_{L_i} \leq g_i(\mathbf{x}) \leq g_{U_i}$, as well as nonlinear equality constraints of the form $h_j(\mathbf{x}) = h_{t_j}$. Some optimizers (e.g., `npsol_`, `optpp_`, `soga`, and `moga` methods) can handle these constraint forms directly, whereas other optimizers (e.g., `asynch_pattern_search`, `dot_`, and `conmin_`, `mesh_adaptive_search`) require Dakota to perform

an internal conversion of all constraints to one-sided inequality constraints of the form $g_i(\mathbf{x}) \leq 0$. In the latter case, the two-sided inequality constraints are treated as $g_i(\mathbf{x}) - g_{U_i} \leq 0$ and $g_{L_i} - g_i(\mathbf{x}) \leq 0$ and the equality constraints are treated as $h_j(\mathbf{x}) - h_{t_j} \leq 0$ and $h_{t_j} - h_j(\mathbf{x}) \leq 0$. The situation is similar for linear constraints: `asynch_`, `pattern_search`, `npsol_`, `optpp_`, `soga`, and `moga` methods support them directly, whereas `dot_` and `conmin_` methods do not. For linear inequalities of the form $a_{L_i} \leq \mathbf{a}_i^T \mathbf{x} \leq a_{U_i}$ and linear equalities of the form $\mathbf{a}_i^T \mathbf{x} = a_{t_j}$, the nonlinear constraint arrays in `dot_` and `conmin_` methods are further augmented to include $\mathbf{a}_i^T \mathbf{x} - a_{U_i} \leq 0$ and $a_{L_i} - \mathbf{a}_i^T \mathbf{x} \leq 0$ in the inequality case and $\mathbf{a}_i^T \mathbf{x} - a_{t_j} \leq 0$ and $a_{t_j} - \mathbf{a}_i^T \mathbf{x} \leq 0$ in the equality case. Awareness of these constraint augmentation procedures can be important for understanding the diagnostic data returned from the `dot_` and `conmin_` methods. Other optimizers fall somewhere in between. `nlpql_` methods support nonlinear equality constraints $h_j(\mathbf{x}) = 0$ and nonlinear one-sided inequalities $g_i(\mathbf{x}) \geq 0$, but does not natively support linear constraints. Constraint mappings are used with NLPQL for both linear and nonlinear cases. Most `coliny_` methods now support two-sided nonlinear inequality constraints and nonlinear constraints with targets, but do not natively support linear constraints.

When gradient and Hessian information is used in the optimization, derivative components are most commonly computed with respect to the active continuous variables, which in this case are the *continuous design variables*. This differs from parameter study methods (for which all continuous variables are active) and from nondeterministic analysis methods (for which the uncertain variables are active). Refer to Chapter 11 of [5] for additional information on derivative components and active continuous variables.

6.8.14.2 Optimizing with Dakota: Choosing a Method

This section summarizes the optimization methods available in Dakota. We group them according to search method and search goal and establish their relevance to types of problems. For a summary of this discussion, see Section 6.4 of [5].

6.8.14.2.1 Gradient-Based Local Methods

Gradient-based optimizers are best suited for efficient navigation to a local minimum in the vicinity of the initial point. They are not intended to find global optima in nonconvex design spaces. For global optimization methods, see Section 6.2.3 of [5]. Gradient-based optimization methods are highly efficient, with the best convergence rates of all of the local optimization methods, and are the methods of choice when the problem is smooth, unimodal, and well-behaved. However, these methods can be among the least robust when a problem exhibits nonsmooth, discontinuous, or multimodal behavior. The derivative-free methods described in Section 6.2.2 of [5] are more appropriate for problems with these characteristics.

Gradient accuracy is a critical factor for gradient-based optimizers, as inaccurate derivatives will often lead to failures in the search or pre-mature termination of the method. Analytic gradients and Hessians are ideal but often unavailable. If analytic gradient and Hessian information can be provided by an application code, a full Newton method will achieve quadratic convergence rates near the solution. If only gradient information is available and the Hessian information is approximated from an accumulation of gradient data, the superlinear convergence rates can be obtained. It is most often the case for engineering applications, however, that a finite difference method will be used by the optimization algorithm to estimate gradient values. Dakota allows the user to select the step size for these calculations, as well as choose between forward-difference and central-difference algorithms. The finite difference step size should be selected as small as possible, to allow for local accuracy and convergence, but not so small that the steps are “in the noise.” This requires an assessment of the local smoothness of the response functions using, for example, a parameter study method. Central differencing will generally produce more reliable gradients than forward differencing but at roughly twice the expense.

Gradient-based methods for nonlinear optimization problems can be described as iterative processes in which a sequence of subproblems, usually which involve an approximation to the full nonlinear problem, are solved until the solution converges to a local optimum of the full problem. The optimization methods available in Dakota fall into several categories, each of which is characterized by the nature of the subproblems solved at each iteration.

Related Topics

- [local_optimization_methods](#)
- [global_optimization_methods](#)

- [bayesian_calibration](#)
- [nonlinear_least_squares](#)
- [advanced_optimization](#)

Related Keywords

- [dl_solver](#) : (Experimental) Dynamically-loaded solver

6.8.15 local_optimization_methods

Description

empty

Related Topics

- [unconstrained](#)
- [constrained](#)
- [sequential_quadratic_programming](#)

Related Keywords

- [coliny_cobyla](#) : Constrained Optimization BY Linear Approximations (COBYLA)
- [nlpql_sqp](#) : NLPQL Sequential Quadratic Program
- [nonlinear_cg](#) : (Experimental) nonlinear conjugate gradient optimization
- [npsol_sqp](#) : NPSOL Sequential Quadratic Program
- [optpp_cg](#) : A conjugate gradient optimization method
- [optpp_fd_newton](#) : Finite Difference Newton optimization method
- [optpp_g_newton](#) : Newton method based least-squares calibration
- [optpp_newton](#) : Newton method based optimization
- [optpp_q_newton](#) : Quasi-Newton optimization method
- [rol](#) : Rapid Optimization Library (ROL) is a large-scale optimization package within Trilinos.

6.8.15.1 unconstrained

Description

empty

Related Topics

Related Keywords

6.8.15.2 `constrained`

Description

empty

Related Topics

Related Keywords

- [coliny_cobyla](#) : Constrained Optimization BY Linear Approximations (COBYLA)

6.8.15.3 `sequential_quadratic_programming`

Description

Sequential Quadratic Programming (SQP) algorithms are a class of mathematical programming problems used to solve nonlinear optimization problems with nonlinear constraints. These methods are a generalization of Newton's method: each iteration involves minimizing a quadratic model of the problem. These subproblems are formulated as minimizing a quadratic approximation of the Lagrangian subject to linearized constraints. Only gradient information is required; Hessians are approximated by low-rank updates defined by the step taken at each iteration. It is important to note that while the solution found by an SQP method will respect the constraints, the intermediate iterates may not. SQP methods available in Dakota are `dot_sqp`, `nlpql_sqp`, `nlsol_sqp`, and `npsol_sqp`. The particular implementation in `nlpql_sqp` uses a variant with distributed and non-monotone line search. Thus, this variant is designed to be more robust in the presence of inaccurate or noisy gradients common in many engineering applications.

Related Topics

Related Keywords

- [nlpql_sqp](#) : NLPQL Sequential Quadratic Program
- [nlsol_sqp](#) : Sequential Quadratic Program for nonlinear least squares
- [npsol_sqp](#) : NPSOL Sequential Quadratic Program

6.8.16 `global_optimization_methods`

Description

empty

Related Topics

Related Keywords

- [asynch_pattern_search](#) : Pattern search, derivative free optimization method
- [coliny_direct](#) : Dividing RECTangles method

- [coliny_ea](#) : Evolutionary Algorithm
- [coliny_pattern_search](#) : Pattern search, derivative free optimization method
- [efficient_global](#) : Global Surrogate Based Optimization, a.k.a. EGO
- [ncsu_direct](#) : Dividing RECTangles method
- [soga](#) : Single-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)

6.8.17 bayesian_calibration

Description

See the discussion of Bayesian Calibration in the Dakota User's Manual [5].

Related Topics

Related Keywords

- [bayes_calibration](#) : Bayesian calibration
- [dream](#) : DREAM (DiffeRential Evolution Adaptive Metropolis)
- [chains](#) : Number of chains in DREAM
- [crossover_chain_pairs](#) : Number of chains used in crossover.
- [gr_threshold](#) : Convergence tolerance for the Gelman-Rubin statistic
- [jump_step](#) : Number of generations a long jump step is taken
- [num_cr](#) : Number of candidate points for each crossover.
- [gpmsa](#) : (Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Bayesian calibration
- [adaptive_metropolis](#) : Use the Adaptive Metropolis MCMC algorithm
- [delayed_rejection](#) : Use the Delayed Rejection MCMC algorithm
- [dram](#) : Use the DRAM MCMC algorithm
- [metropolis_hastings](#) : Use the Metropolis-Hastings MCMC algorithm
- [proposal_covariance](#) : Defines the technique used to generate the MCMC proposal covariance.
- [derivatives](#) : Use derivatives to inform the MCMC proposal covariance.
- [prior](#) : Uses the covariance of the prior distributions to define the MCMC proposal covariance.
- [muq](#) : Markov Chain Monte Carlo algorithms from the MUQ package
- [adaptive_metropolis](#) : Use the Adaptive Metropolis MCMC algorithm
- [delayed_rejection](#) : Use the Delayed Rejection MCMC algorithm
- [dram](#) : Use the DRAM MCMC algorithm
- [metropolis_hastings](#) : Use the Metropolis-Hastings MCMC algorithm
- [proposal_covariance](#) : Defines the technique used to generate the MCMC proposal covariance.
- [derivatives](#) : Use derivatives to inform the MCMC proposal covariance.
- [prior](#) : Uses the covariance of the prior distributions to define the MCMC proposal covariance.

- [queso](#) : Markov Chain Monte Carlo algorithms from the QUESO package
- [adaptive_metropolis](#) : Use the Adaptive Metropolis MCMC algorithm
- [delayed_rejection](#) : Use the Delayed Rejection MCMC algorithm
- [dram](#) : Use the DRAM MCMC algorithm
- [metropolis_hastings](#) : Use the Metropolis-Hastings MCMC algorithm
- [multilevel](#) : Use the multilevel MCMC algorithm.
- [proposal_covariance](#) : Defines the technique used to generate the MCMC proposal covariance.
- [derivatives](#) : Use derivatives to inform the MCMC proposal covariance.
- [prior](#) : Uses the covariance of the prior distributions to define the MCMC proposal covariance.

6.8.18 nonlinear_least_squares

Description

Dakota's least squares branch currently contains three methods for solving nonlinear least squares problems:

- NL2SOL, a trust-region method that adaptively chooses between two Hessian approximations (Gauss-Newton and Gauss-Newton plus a quasi-Newton approximation to the rest of the Hessian)
- NLSSOL, a sequential quadratic programming (SQP) approach that is from the same algorithm family as NPSOL
- Gauss-Newton, which supplies the Gauss-Newton Hessian approximation to the full-Newton optimizers from OPT++.

The important difference of these algorithms from general-purpose optimization methods is that the response set is defined by calibration terms (e.g. separate terms for each residual), rather than an objective function. Thus, a finer granularity of data is used by least squares solvers as compared to that used by optimizers. This allows the exploitation of the special structure provided by a sum of squares objective function.

Related Topics

Related Keywords

- [nl2sol](#) : Trust-region method for nonlinear least squares
- [nlssol_sqp](#) : Sequential Quadratic Program for nonlinear least squares

6.8.19 advanced_optimization

Description

empty

Related Topics

- [scaling](#)
- [multiobjective_methods](#)
- [surrogate_based_optimization_methods](#)

Related Keywords

6.8.19.1 scaling

Description

empty

Related Topics

Related Keywords

6.8.19.2 multiobjective_methods

Description

empty

Related Topics

Related Keywords

6.8.19.3 surrogate_based_optimization_methods

Description

empty

Related Topics

Related Keywords

- [efficient_global](#) : Global Surrogate Based Optimization, a.k.a. EGO
- [surrogate_based_global](#) : Adaptive Global Surrogate-Based Optimization
- [surrogate_based_local](#) : Local Surrogate Based Optimization

6.9 advanced_topics

Description

Advanced Dakota capabilities

Related Topics

- [advanced_strategies](#)
- [advanced_model_recursion](#)
- [advanced_simulation_interfaces](#)
- [advanced_optimization](#)

Related Keywords

6.9.1 advanced_strategies

Description

empty

Related Topics

Related Keywords

6.9.2 advanced_model_recursion

Description

empty

Related Topics

- [hybrid_and_recursions_logic](#)

Related Keywords

6.9.3 hybrid_and_recursions_logic

Description

empty

Related Topics

Related Keywords

6.9.4 advanced_simulation_interfaces

Description

empty

Related Topics

- [simulation_failure](#)
- [concurrency_and_parallelism](#)

Related Keywords

6.9.5 simulation_failure

Description

empty

Related Topics

Related Keywords

6.9.6 concurrency_and_parallelism

Description

empty

Related Topics

Related Keywords

- [processors_per_analysis](#) : Specify the number of processors per analysis when Dakota is run in parallel
- [analysis_scheduling](#) : Specify the scheduling of concurrent analyses when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel analysis scheduling
- [peer](#) : Specify a peer partition for parallel analysis scheduling
- [analysis_servers](#) : Specify the number of analysis servers when Dakota is run in parallel
- [asynchronous](#) : Specify local evaluation or analysis concurrency
- [analysis_concurrency](#) : Limit the number of analysis drivers within an evaluation that Dakota will schedule
- [evaluation_concurrency](#) : Determine how many concurrent evaluations Dakota will schedule
- [local_evaluation_scheduling](#) : Control how local asynchronous jobs are scheduled
- [batch](#) : Perform evaluations in batches
- [size](#) : Limit the number of evaluations in a batch
- [master](#) : Specify a dedicated master partition for parallel evaluation scheduling
- [peer](#) : Specify a peer partition for parallel evaluation scheduling
- [dynamic](#) : Specify dynamic scheduling in a peer partition when Dakota is run in parallel.
- [static](#) : Specify static scheduling in a peer partition when Dakota is run in parallel.
- [evaluation_servers](#) : Specify the number of evaluation servers when Dakota is run in parallel
- [processors_per_evaluation](#) : Specify the number of processors per evaluation server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel

- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel
- [iterator_scheduling](#) : Specify the scheduling of concurrent iterators when Dakota is run in parallel
- [master](#) : Specify a dedicated master partition for parallel iterator scheduling
- [peer](#) : Specify a peer partition for parallel iterator scheduling
- [iterator_servers](#) : Specify the number of iterator servers when Dakota is run in parallel
- [processors_per_iterator](#) : Specify the number of processors per iterator server when Dakota is run in parallel

6.9.7 advanced_optimization

Description

empty

Related Topics

- [scaling](#)
- [multiobjective_methods](#)
- [surrogate_based_optimization_methods](#)

Related Keywords

6.9.7.1 scaling

Description

empty

Related Topics

Related Keywords

6.9.7.2 multiobjective_methods

Description

empty

Related Topics

Related Keywords

6.9.7.3 surrogate_based_optimization_methods

Description

empty

Related Topics

Related Keywords

- [efficient_global](#) : Global Surrogate Based Optimization, a.k.a. EGO
- [surrogate_based_global](#) : Adaptive Global Surrogate-Based Optimization
- [surrogate_based_local](#) : Local Surrogate Based Optimization

6.10 packages

Description

This topic organizes information about the different software packages (libraries) that are integrated into Dakota

Related Topics

- [package_coliny](#)
- [package_conmin](#)
- [package_ddace](#)
- [package_dot](#)

- [package_fsudace](#)
- [package_hopspack](#)
- [package_jega](#)
- [package_nlpql](#)
- [package_npsol](#)
- [package_optpp](#)
- [package_psuade](#)
- [package_queso](#)
- [package_scolib](#)

Related Keywords

6.10.1 package_coliny

Description

SCOLIB (formerly known as COLINY) is a collection of nongradient-based optimizers that support the Common Optimization Library Interface (COLIN). SCOLIB optimizers currently include `coliny_cobyala`, `coliny_direct`, `coliny_ea`, `coliny_pattern_search` and `coliny_solis_wets`. (Yes, the input spec still has "coliny" prepended to the method name.) Additional SCOLIB information is available from <https://software.sandia.gov/trac/acro>.

SCOLIB solvers now support bound constraints and general nonlinear constraints. Supported nonlinear constraints include both equality and two-sided inequality constraints. SCOLIB solvers do not yet support linear constraints. Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. Specific exceptions to this method for handling constraint violations are noted below. (The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.)

The method independent controls for `max_iterations` and `max_function_evaluations` limit the number of major iterations and the number of function evaluations that can be performed during a SCOLIB optimization, respectively. The `convergence_tolerance` control defines the threshold value on relative change in the objective function that indicates convergence. The `output_verbosity` specification controls the amount of information generated by SCOLIB: the `silent`, `quiet`, and `normal` settings correspond to minimal reporting from SCOLIB, whereas the `verbose` setting corresponds to a higher level of information, and `debug` outputs method initialization and a variety of internal SCOLIB diagnostics. The majority of SCOLIB's methods perform independent function evaluations that can directly take advantage of Dakota's parallel capabilities. Only `coliny_solis_wets`, `coliny_cobyala`, and certain configurations of `coliny_pattern_search` are inherently serial. The parallel methods automatically utilize parallel logic when the Dakota configuration supports parallelism. Lastly, neither `speculative` gradients nor linear constraints are currently supported with SCOLIB.

Some SCOLIB methods exploit parallelism through the use of Dakota's concurrent function evaluations. The nature of the algorithms, however, limits the amount of concurrency that can be exploited. The maximum amount of evaluation concurrency that can be leveraged by the various methods is as follows:

- COBYLA: one
- DIRECT: twice the number of variables
- Evolutionary Algorithms: size of the population
- Pattern Search: size of the search pattern
- Solis-Wets: one

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

Each of the SCOLIB methods supports the `solution_target` control, which defines a convergence criterion in which the optimizer will terminate if it finds an objective function value lower than the specified target.

Related Topics

Related Keywords

- [coliny_beta](#) : (Experimental) Coliny beta solver
- [coliny_cobyta](#) : Constrained Optimization BY Linear Approximations (COBYLA)
- [coliny_direct](#) : Dividing RECTangles method
- [coliny_ea](#) : Evolutionary Algorithm
- [coliny_pattern_search](#) : Pattern search, derivative free optimization method
- [coliny_solis_wets](#) : Simple greedy local search method

6.10.2 package_conmin

Description

The CONMIN library [86] is a public domain library of nonlinear programming optimizers, specifically the Fletcher-Reeves conjugate gradient ([conmin_frcg](#)) method for unconstrained optimization, and the method of feasible directions ([conmin_mfd](#)) for constrained optimization. As CONMIN was a predecessor to the DOT commercial library, the algorithm controls are very similar.

Related Topics

Related Keywords

- [conmin_mfd](#) : CONMIN method of feasible directions

6.10.3 package_ddace

Description

The Distributed Design and Analysis of Computer Experiments (DDACE) library provides the following DACE techniques: grid sampling ([grid](#)), pure random sampling ([random](#)), orthogonal array sampling ([oas](#)), latin hypercube sampling ([lhs](#)), orthogonal array latin hypercube sampling ([oa_lhs](#)), Box-Behnken ([box_behnken](#)), and central composite design ([central_composite](#)).

It is worth noting that there is some overlap in sampling techniques with those available from the nondeterministic branch. The current distinction is that the nondeterministic branch methods are designed to sample within a variety of probability distributions for uncertain variables, whereas the design of experiments methods treat all variables as having uniform distributions. As such, the design of experiments methods are well-suited for performing parametric studies and for generating data sets used in building global approximations, but are not currently suited for

assessing the effect of uncertainties characterized with probability distribution. If a design of experiments over both design/state variables (treated as uniform) and uncertain variables (with probability distributions) is desired, then sampling can support this with `active all` specified in the Variables specification block.

Related Topics

Related Keywords

- [dace](#) : Design and Analysis of Computer Experiments

6.10.4 package_dot

Description

The DOT library [88] contains nonlinear programming optimizers, specifically the Broyden-Fletcher-Goldfarb--Shanno ([dot_bfgs](#)) and Fletcher-Reeves conjugate gradient ([dot_frcg](#)) methods for unconstrained optimization, and the modified method of feasible directions ([dot_mmfd](#)), sequential linear programming ([dot_slp](#)), and sequential quadratic programming ([dot_sqp](#)) methods for constrained optimization. To use DOT, one of these methods must be specified.

Specialized handling of linear constraints is supported with DOT; linear constraint coefficients, bounds, and targets can be provided to DOT at start-up and tracked internally.

Method Independent Controls - Stopping Criteria

Stopping criteria are set by: `max_iterations`, `max_function_evaluations`, `convergence_tolerance`, and `constraint_tolerance`

Note: The `convergence_tolerance` criterion must be satisfied for two consecutive iterations before DOT will terminate.

Method Independent Controls - Output

The output verbosity specification controls the amount of information generated by DOT: the `silent` and `quiet` settings result in header information, final results, and objective function, constraint, and parameter information on each iteration; whereas the `verbose` and `debug` settings add additional information on gradients, search direction, one-dimensional search results, and parameter scaling factors.

Concurrency

DOT contains no parallel algorithms which can directly take advantage of concurrent evaluations. However, if `numerical_gradients` with `method_source dakota` is specified, then the finite difference function evaluations can be performed concurrently (using any of the parallel modes described in the Users Manual [5]). In addition, if `speculative` is specified, then gradients (`dakota numerical` or `analytic` gradients) will be computed on each line search evaluation in order to balance the load and lower the total run time in parallel optimization studies.

Related Topics

Related Keywords

- [dot_sqp](#) : DOT Sequential Quadratic Program

6.10.5 package_fsudace

Description

The Florida State University Design and Analysis of Computer Experiments (FSUDace) library provides the following DACE techniques: quasi-Monte Carlo sampling (`fsu_quasi_mc`) based on the Halton sequence (`halton`) or the Hammersley sequence (`hammersley`), and Centroidal Voronoi Tessellation (`fsu_cvt`).

Related Topics

Related Keywords

- [quality_metrics](#) : Calculate metrics to assess the quality of quasi-Monte Carlo samples
- [fsu_cvt](#) : Design of Computer Experiments - Centroidal Voronoi Tessellation
- [quality_metrics](#) : Calculate metrics to assess the quality of quasi-Monte Carlo samples
- [halton](#) : Generate samples from a Halton sequence
- [fsu_quasi_mc](#) : Design of Computer Experiments - Quasi-Monte Carlo sampling
- [halton](#) : Generate samples from a Halton sequence
- [hammersley](#) : Use Hammersley sequences
- [quality_metrics](#) : Calculate metrics to assess the quality of quasi-Monte Carlo samples

6.10.6 package_hopspack

Description

The HOPSPACK software [72] contains the asynchronous parallel pattern search (APPS) algorithm [38]. It can handle unconstrained problems as well as those with bound constraints, linear constraints, and general nonlinear constraints.

HOPSPACK is available to the public under the GNU LGPL and the source code is included with Dakota. HOPSPACK-specific software documentation is available from <https://software.sandia.gov/trac/hopspack>.

Related Topics

Related Keywords

- [asynch_pattern_search](#) : Pattern search, derivative free optimization method

6.10.7 package_jega

Description

The JEGA library [19] contains two global optimization methods. The first is a Multi-objective Genetic Algorithm (MOGA) which performs Pareto optimization. The second is a Single-objective Genetic Algorithm (SOGA) which performs optimization on a single objective function. Both methods support general constraints and a mixture of real and discrete variables. The JEGA library was written by John Eddy, currently a member of the technical staff in the System Readiness and Sustainment Technologies department at Sandia National Laboratories in Albuquerque. These algorithms are accessed as `moga` and `soga` within Dakota.

Related Topics

Related Keywords

- [moga](#) : Multi-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)
- [soga](#) : Single-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)

6.10.8 package_nlpql

Description

The NLPQL library includes a sequential quadratic programming (SQP) optimizer, specified as Dakota's `nlpql__sqp` method, for constrained optimization. The particular implementation used is NLPQLP [77], a variant with distributed and non-monotone line search. NLPQL is commercially licensed (available from the author) and not distributed with public versions of Dakota.

Related Topics

Related Keywords

- [nlpql_sqp](#) : NLPQL Sequential Quadratic Program

6.10.9 package_npsol

Description

The NPSOL library [33] contains a sequential quadratic programming (SQP) implementation (the `npsol_sqp` method). SQP is a nonlinear programming optimizer for constrained minimization.

Related Topics

Related Keywords

- [npsol_sqp](#) : NPSOL Sequential Quadratic Program

6.10.10 package_optpp

Description

The OPT++ library [62] contains primarily gradient-based nonlinear programming optimizers for unconstrained, bound-constrained, and nonlinearly constrained minimization: Polak-Ribiere conjugate gradient (Dakota's `optpp__cg` method), quasi-Newton (Dakota's `optpp_q_newton` method), finite difference Newton (Dakota's `optpp__fd_newton` method), and full Newton (Dakota's `optpp_newton` method).

The conjugate gradient method is strictly unconstrained, and each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a nonlinear interior-point approach to manage the constraints. The library also contains a direct search algorithm, PDS (parallel direct search, Dakota's `optpp_pds` method), which supports bound constraints.

Controls

1. `max_iterations`

2. `max_function_evaluations`
3. `convergence_tolerance`
4. `output`
5. `speculative`

Concurrency

OPT++'s gradient-based methods are not parallel algorithms and cannot directly take advantage of concurrent function evaluations. However, if `numerical_gradients` with `method_source dakota` is specified, a parallel Dakota configuration can utilize concurrent evaluations for the finite difference gradient computations.

Constraints

Linear constraint specifications are supported by each of the Newton methods (`optpp_newton`, `optpp_q_newton`, `optpp_fd_newton`, and `optpp_g_newton`)

`optpp_cg` must be unconstrained

`optpp_pds` can be, at most, bound-constrained.

Related Topics

Related Keywords

- [optpp_cg](#) : A conjugate gradient optimization method
- [optpp_fd_newton](#) : Finite Difference Newton optimization method
- [optpp_g_newton](#) : Newton method based least-squares calibration
- [optpp_newton](#) : Newton method based optimization
- [optpp_pds](#) : Simplex-based derivative free optimization method
- [optpp_q_newton](#) : Quasi-Newton optimization method

6.10.11 package_psuade

Description

The Problem Solving Environment for Uncertainty Analysis and Design Exploration (PSUADE) is a Lawrence Livermore National Laboratory tool for metamodeling, sensitivity analysis, uncertainty quantification, and optimization. Its features include non-intrusive and parallel function evaluations, sampling and analysis methods, an integrated design and analysis framework, global optimization, numerical integration, response surfaces (MARS and higher order regressions), graphical output with Pgplot or Matlab, and fault tolerance [84].

Related Topics

Related Keywords

- [psuade_moat](#) : Morris One-at-a-Time

6.10.12 package_queso

Description

QUESO stands for Quantification of Uncertainty for Estimation, Simulation, and Optimization. It supports Bayesian calibration methods. It is developed at The University of Texas at Austin.

Related Topics

Related Keywords

- [bayes_calibration](#) : Bayesian calibration
- [gpmsa](#) : (Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Bayesian calibration
- [queso](#) : Markov Chain Monte Carlo algorithms from the QUESO package

6.10.13 package_scolib

Description

SCOLIB (formerly known as COLINY) is a collection of nongradient-based optimizers that support the Common Optimization Library Interface (COLIN). SCOLIB optimizers currently include `coliny_cobyala`, `coliny_direct`, `coliny_ea`, `coliny_pattern_search` and `coliny_solis_wets`. (Yes, the input spec still has "coliny" prepended to the method name.) Additional SCOLIB information is available from <https://software.sandia.gov/trac/acro>.

SCOLIB solvers now support bound constraints and general nonlinear constraints. Supported nonlinear constraints include both equality and two-sided inequality constraints. SCOLIB solvers do not yet support linear constraints. Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. Specific exceptions to this method for handling constraint violations are noted below. (The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.)

The method independent controls for `max_iterations` and `max_function_evaluations` limit the number of major iterations and the number of function evaluations that can be performed during a SCOLIB optimization, respectively. The `convergence_tolerance` control defines the threshold value on relative change in the objective function that indicates convergence. The `output_verbosity` specification controls the amount of information generated by SCOLIB: the `silent`, `quiet`, and `normal` settings correspond to minimal reporting from SCOLIB, whereas the `verbose` setting corresponds to a higher level of information, and `debug` outputs method initialization and a variety of internal SCOLIB diagnostics. The majority of SCOLIB's methods perform independent function evaluations that can directly take advantage of Dakota's parallel capabilities. Only `coliny_solis_wets`, `coliny_cobyala`, and certain configurations of `coliny_pattern_search` are inherently serial. The parallel methods automatically utilize parallel logic when the Dakota configuration supports parallelism. Lastly, neither `speculative` gradients nor linear constraints are currently supported with SCOLIB.

Some SCOLIB methods exploit parallelism through the use of Dakota's concurrent function evaluations. The nature of the algorithms, however, limits the amount of concurrency that can be exploited. The maximum amount of evaluation concurrency that can be leveraged by the various methods is as follows:

- COBYLA: one
- DIRECT: twice the number of variables
- Evolutionary Algorithms: size of the population
- Pattern Search: size of the search pattern
- Solis-Wets: one

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

Each of the SCOLIB methods supports the `solution_target` control, which defines a convergence criterion in which the optimizer will terminate if it finds an objective function value lower than the specified target.

Related Topics

Related Keywords

- [coliny_beta](#) : (Experimental) Coliny beta solver
- [coliny_cobyla](#) : Constrained Optimization BY Linear Approximations (COBYLA)
- [coliny_direct](#) : Dividing RECTangles method
- [coliny_ea](#) : Evolutionary Algorithm
- [coliny_pattern_search](#) : Pattern search, derivative free optimization method
- [coliny_solis_wets](#) : Simple greedy local search method

Chapter 7

Keywords Area

This page summarizes the overall input file structure, syntax, and the six types of blocks that may appear in Dakota input. Some are optional and some may appear multiple times:

- [environment](#) (optional; 0 or 1 may appear)
- [method](#) (required; 1 or more may appear)
- [model](#) (optional; 0 or more may appear)
- [variables](#) (required; 1 or more may appear)
- [interface](#) (required; 1 or more may appear)
- [responses](#) (required; 1 or more may appear)

Introduction to Dakota Keywords

In Dakota, the *environment* manages execution modes and I/O streams and defines the top-level iterator. Generally speaking, an iterator contains a model and a model contains a set of *variables*, an *interface*, and a set of *responses*. An iterator repeatedly operates on the model to map the variables into responses using the interface. Each of these six components (environment, method, model, variables, interface, and responses) are separate specifications in the user's input file, and as a whole, determine the study to be performed during an execution of the Dakota software.

A Dakota execution is limited to a single environment, but may involve multiple methods and multiple models. In particular, advanced iterators (i.e., meta- and component-based iterators) and advanced models (i.e., nested and surrogate models) may specialize to include recursions with additional sub-iterators and sub-models. Since each model may contain its own variables, interface, and responses, there may be multiple specifications of the method, model, variables, interface, and responses sections.

Keyword Pages

Every Dakota keyword has its own page in this manual. The page describes:

- Whether the keyword takes `ARGUMENTS`, and the data type Additional notes about `ARGUMENTS` can be found here: [Specifying Arguments](#).
- Whether it has an `ALIAS`
- Which additional keywords can be specified to change its behavior
- Which of these additional keywords are required or optional
- Additional information about how to use the keyword in an input file

7.1 environment

- [Keywords Area](#)
- [environment](#)

Top-level settings for Dakota execution

Topics

This keyword is related to the topics:

- [block](#)

Specification

Alias: none

Argument(s): none

Default: no environment

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		tabular_data	Write a tabular results file with variable and response history
	Optional		output_file	Base filename for output redirection
	Optional		error_file	Base filename for error redirection
	Optional		read_restart	Base filename for restart file read
	Optional		write_restart	Base filename for restart file write
	Optional		output_precision	Control the output precision
	Optional		results_output	(Experimental) Write a summary file containing the final results

	Optional	graphics	(DEPRECATED) Display plots of variables and responses
	Optional	check	Invoke Dakota in input check mode
	Optional	pre_run	Invoke Dakota with pre-run mode active
	Optional	run	Invoke Dakota with run mode active
	Optional	post_run	Invoke Dakota with post-run mode active
	Optional	top_method_pointer	Identify which method leads the Dakota study

Description

The environment section in a Dakota input file is optional. It specifies the top-level solution environment, optionally including run modes, output controls, and identification of the primary iterative method (`top_method_pointer`). The output-related keywords address generation of tabular and results data, and precision of numerical output.

Run Mode Defaults

Dakota run phases include `check`, `pre_run`, `run`, and `post_run`. The default behavior is to `pre_run`, `run`, and `post_run`, though any or all of these may be specified to select specific run phases. Specifying `check` will cause Dakota to exit before any selected run modes.

7.1.1 tabular_data

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)

Write a tabular results file with variable and response history

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: `tabular_graphics_data`

Argument(s): none

Default: no tabular data output

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		tabular_data_file	File name for tabular data output
	Optional (<i>Choose One</i>)	Tabular Data Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifying the `tabular_data` flag writes to a data file the same variable and response function history data plotted when using the (deprecated) `graphics` flag. Within the generated data file, the variables and response functions appear as columns and each function evaluation provides a new table row. This capability is most useful for post-processing of Dakota results with third-party graphics tools such as MatLab, Excel, Tecplot, etc.

There is no dependence between the `graphics` flag and the `tabular_data` flag; they may be used independently or concurrently.

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

See Also

These keywords may also be of interest:

- [graphics](#)

7.1.1.1 `tabular_data_file`

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)
- [tabular_data_file](#)

File name for tabular data output

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: tabular_graphics_file

Argument(s): STRING

Default: dakota_tabular.dat

Description

Specifies a name to use for the tabular data file, overriding the default `dakota_tabular.dat`.

7.1.1.2 custom_annotated

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) annotated.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.1.1.2.1 header

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.1.1.2.2 `eval_id`

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.1.1.2.3 `interface_id`

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.1.1.3 annotated

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.1.1.4 freeform

- [Keywords Area](#)
- [environment](#)
- [tabular_data](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.1.2 output_file

- [Keywords Area](#)
- [environment](#)
- [output_file](#)

Base filename for output redirection

Topics

This keyword is related to the topics:

- [dakota_IO](#)
- [command_line_options](#)

Specification

Alias: none

Argument(s): STRING

Default: output to console, not file

Description

Specify a base filename to which Dakota output will be directed. Output will (necessarily) be redirected after the input file is parsed. This option is overridden by any command-line `-output` option.

Default Behavior

Output to console (screen).

7.1.3 error_file

- [Keywords Area](#)
- [environment](#)
- [error_file](#)

Base filename for error redirection

Topics

This keyword is related to the topics:

- [dakota_IO](#)
- [command_line_options](#)

Specification

Alias: none

Argument(s): STRING

Default: errors to console, not file

Description

Specify a base filename to which Dakota errors will be directed. Errors will (necessarily) be redirected after the input file is parsed. This option is overridden by any command-line `-error` option.

Default Behavior

Errors to console (screen).

7.1.4 read_restart

- [Keywords Area](#)
- [environment](#)
- [read_restart](#)

Base filename for restart file read

Topics

This keyword is related to the topics:

- [dakota_IO](#)
- [command_line_options](#)

Specification**Alias:** none**Argument(s):** STRING**Default:** no restart read**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		stop_restart	Restart record number at which to stop reading the restart file.

Description

Specify a base filename for the restart file Dakota should read. This option is overridden by any command-line `-read_restart` option.

Default Behavior

No restart file is read.

7.1.4.1 `stop_restart`

- [Keywords Area](#)
- [environment](#)
- [read_restart](#)
- [stop_restart](#)

Restart record number at which to stop reading the restart file.

Topics

This keyword is related to the topics:

- [dakota_IO](#)

Specification

Alias: none

Argument(s): INTEGER

Default: read all records

Description

This option is overridden by any command-line `-stop_restart` option.

7.1.5 `write_restart`

- [Keywords Area](#)
- [environment](#)
- [write_restart](#)

Base filename for restart file write

Topics

This keyword is related to the topics:

- [dakota_IO](#)
- [command_line_options](#)

Specification

Alias: none

Argument(s): STRING

Default: dakota.rst

Description

Specify a base filename for the restart file Dakota should write. This option is overridden by any command-line `-write_restart` option.

7.1.6 `output_precision`

- [Keywords Area](#)
- [environment](#)
- [output_precision](#)

Control the output precision

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 10

Description

The precision of numeric output precision can be set with `output_precision`, with an upper limit of 16. When not specified, most Dakota output will default to a precision of 10, though filesystem interfaces and pre-run output use higher precision for accuracy and better results reproducibility.

7.1.7 results_output

- [Keywords Area](#)
- [environment](#)
- [results_output](#)

(Experimental) Write a summary file containing the final results

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: no results output

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		results_output_file	The base file name of the results file
	Optional		text	Write results to file in text format
	Optional		hdf5	Write results to file in HDF5 format

Description

(Experimental)

Dakota writes final results for most methods to the screen (console). This keyword enables experimental support for writing them to disk, as well. By default, they are written in a structured text format to a file named `dakota_results.txt`. Text format can be explicitly selected using the `text` sub-keyword. If Dakota was built with HDF5 enabled, the sub-keyword `hdf5` causes results to be written in HDF5 format, as well.

The name of the results file can be changed using the `results_output_file` keyword.

Text output eventually will be deprecated, and HDF5 will be enabled in all distributed builds of Dakota. The layout of the HDF5 file is described in the [Dakota HDF5 Output](#) section of this manual. The contents, organization, and format of results files are all under active development and are subject to change.

7.1.7.1 results_output_file

- [Keywords Area](#)
- [environment](#)
- [results_output](#)

- [results_output_file](#)

The base file name of the results file

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): STRING

Default: dakota_results

Description

The default filename is `dakota_results.txt` for text output and `dakota_results.h5` for HDF5. The `dakota_results` stem can be changed using this keyword. Do not include the extension; these will be added by Dakota.

7.1.7.2 text

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [text](#)

Write results to file in text format

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: text output

Description

When this keyword is present, Dakota will write (some) final method results to disk in a custom structured text format instead of just to the console. Support for text format results output will be deprecated in a future release, and the contents and organization of the output file is subject to change.

The default name of the file is `dakota_results.txt`. This can be changed using the `results_output_file` keyword.

Examples

```
environment
  results_output
    text
      results_output_file 'my_results' # The .txt extension will be added
```

7.1.7.3 hdf5

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)

Write results to file in HDF5 format

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: no HDF5 output

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		model_selection	Select the models that write evaluation data to HDF5

	Optional	interface_selection	Select the models that write evaluation data to HDF5
--	-----------------	-------------------------------------	--

Description

When this keyword is present, Dakota will right (some) final method results to disk in HDF5 format instead of just to the console. This is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

With `hdf5` selected, the default name of the output file is `dakota_results.h5`. This can be changed using the `results_output_file` keyword.

Examples

```
environment
  results_output
    hdf5
    results_output_file 'my_results' # The .h5 extension will be added
```

7.1.7.3.1 model_selection

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [model_selection](#)

Select the models that write evaluation data to HDF5

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Model Evaluation Storage Selection (Group 1)	top_method	Write evaluation data only for the top-level method's model to HDF5
			none	Write evaluation data for no models to HDF5
			all_methods	Write evaluation data to HDF5 for all models that belong directly to methods
			all	Write evaluation data to HDF5 for all models

Description

By default, when HDF5 output is enabled, Dakota writes evaluation data only for the model that belongs to the top-level method. This keyword group is used to override the default.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

The example below will be used to explain the effect of each keyword.

Examples

```
environment
  results_output
    hdf5
      # model_selection
      # top_method
      # all_methods
      # all
      # none
      results_output_file 'my_results' # The .h5 extension will be added

  method_pointer 'opt'

method
  id_method 'opt'
  optpp_q_newton
  model_pointer 'surr'

model
  id_model 'surr'
  surrogate global gaussian_process surfpack
  dace_method_pointer 'training'

method
  id_method 'training'
  sampling
    seed 1234
    samples 20
  model_pointer 'truth_m'

model
```

```

id_model 'truth_m'
simulation

interface
  id_interface 'truth'
  direct
  analysis_drivers 'text_book'

variables
  continuous_design 2
  descriptors 'x1' 'x2'
  lower_bounds -2.0 -2.0
  upper_bounds 2.0 2.0

responses
  objective_functions 2
  descriptors 'f1' 'f2'
  analytic_gradients
  no_hessians

```

7.1.7.3.1.1 top_method

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [model_selection](#)
- [top_method](#)

Write evaluation data only for the top-level method's model to HDF5

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: top_method

Description

When this option is selected, evaluation data only for the model that belongs to the top-level method will be written to HDF5.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

Examples

For the example input in the parent keyword, the following model groups would be written:

- `/models/recast/RECAST_surr_RECAST_1/`

The recast model, in this case, is automatically generated by Dakota to handle summing the pair of objective functions. It is the model that belongs directly to the method 'opt', which is the top-level method. See [Sources of Evaluation Data](#) for further explanation.

The following links would be added to methods and models `sources` groups:

- `/methods/opt/sources/RECAST_surr_RECAST_1 → /models/recast/RECAST_surr_RECAST_1/`

7.1.7.3.1.2 none

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [model_selection](#)
- [none](#)

Write evaluation data for no models to HDF5

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: top_method

Description

When this option is selected, no model evaluation data is written to HDF5, and the `/models` group itself is not created.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

7.1.7.3.1.3 all_methods

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [model_selection](#)
- [all_methods](#)

Write evaluation data to HDF5 for all models that belong directly to methods

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: top_method

Description

When this option is selection, evaluation data for all models that belong directly to methods will be written to HDF5. Models that belong to other models (e.g. wrapped models in a recast relationship) are not stored.

By default, when HDF5 output is enabled, Dakota writes evaluation data only for the model that belongs to the top-level method. This keyword group is used to override the default.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

Examples

For the example input in the parent keyword, the following model groups would be written:

- /models/recast/RECAST_surr_RECAST_1/
- /models/simulation/truth_m/

The recast model, in this case, is automatically generated by Dakota to handle summing the pair of objective functions. It is the model that belongs directly to the method 'opt'. See [Sources of Evaluation Data](#) for further explanation.

The following links would be added to methods and models *sources* groups:

- /methods/opt/sources/RECAST_surr_RECAST_1 → /models/recast/RECAST_surr_RECAST_1/

- `/methods/training/sources/truth_m` → `/models/simulation/truth_m/`

Depending on [interface_selection](#), the following links may also be added to model `sources` groups:

- `/models/simulation/truth_m/sources/truth` → `/interfaces/truth/truth_m/`

7.1.7.3.1.4 all

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [model_selection](#)
- [all](#)

Write evaluation data to HDF5 for all models

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: `top_method`

Description

When this option is selected, evaluation for all models will be written to HDF5.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

Examples

For the example input in the parent keyword, the following model groups would be written:

- `/models/recast/RECAST_surr_RECAST_1/`
- `/models/simulation/truth_m/`
- `/models/surrogate/surr/`

The recast model, in this case, is automatically generated by Dakota to handle summing the pair of objective functions. See [Sources of Evaluation Data](#) for further explanation.

The following links would be added to methods and models `sources` groups:

- `/methods/opt/sources/RECAST_surr_RECAST_1` → `/models/recast/RECAST_surr_-RECAST_1/`
- `/methods/training/sources/truth_m` → `/models/simulation/truth_m/`
- `/models/recast/RECAST_surr_RECAST_1/sources/surr` → `/models/surrogate/surr/`

Depending on [interface_selection](#), the following links may also be added to model sources groups:

- `/models/surrogate/surr/sources/APPROX_INTERFACE_1` → `/interfaces/APPROX_-INTERFACE_1/surr/`
- `/models/simulation/truth_m/sources/truth` → `/interfaces/truth/truth_m/`

7.1.7.3.2 interface_selection

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [interface_selection](#)

Select the models that write evaluation data to HDF5

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Interface Evaluation Storage Selection (Group 1)	none	Write evaluation data for no interfaces to HDF5
			simulation	Write evaluation data only for simulation interfaces to HDF5

			all	Write evaluation data for all interfaces to HDF5
--	--	--	-----	--

Description

By default, when HDF5 output is enabled, Dakota writes evaluation data for all simulation interfaces (interfaces of type fork, system, direct, etc, but not approximation interfaces, which are constructed by models of type surrogate). This keyword group is used to override the default.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

The example below will be used to explain the effect of each keyword.

Examples

```
environment
  results_output
    hdf5
      # interface_selection
      # simulation
      # all
      # none
      results_output_file 'my_results' # The .h5 extension will be added

  method_pointer 'opt'

method
  id_method 'opt'
  optpp_q_newton
  model_pointer 'surr'

model
  id_model 'surr'
  surrogate global gaussian_process surfpack
  dace_method_pointer 'training'

method
  id_method 'training'
  sampling
    seed 1234
    samples 20
  model_pointer 'truth_m'

model
  id_model 'truth_m'
  simulation

interface
  id_interface 'truth'
  direct
    analysis_drivers 'text_book'

variables
  continuous_design 2
  descriptors 'x1' 'x2'
  lower_bounds -2.0 -2.0
  upper_bounds 2.0 2.0

responses
  objective_functions 2
  descriptors 'f1' 'f2'
  analytic_gradients
  no_hessians
```

7.1.7.3.2.1 none

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [interface_selection](#)
- [none](#)

Write evaluation data for no interfaces to HDF5

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: simulation

Description

When this option is selected, no interface evaluation data is written to HDF5, and the `/interfaces` group is not created.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

7.1.7.3.2.2 simulation

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [interface_selection](#)
- [simulation](#)

Write evaluation data only for simulation interfaces to HDF5

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: simulation

Description

When this option is selected, evaluation data only for simulation interfaces will be written to HDF5. Simulation interfaces include fork, system, direct, matlab, python, scilab, and grid interfaces, but not approximation interfaces, which are automatically generated by Dakota to be used by surrogate models.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

Examples

For the example input in the parent keyword, the following interface groups would be written:

- `/interfaces/truth/truth_m/`

Depending on the [model_selection](#), the following links may be added to model `sources` groups.

- `/models/simulation/truth_m/sources/truth → /interfaces/truth/truth_m/`

7.1.7.3.2.3 all

- [Keywords Area](#)
- [environment](#)
- [results_output](#)
- [hdf5](#)
- [interface_selection](#)
- [all](#)

Write evaluation data for all interfaces to HDF5

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: top_simulation

Description

When this option is selected, evaluation data for all interfaces, including both simulations and approximations, will be written to HDF5.

HDF5 output is an experimental feature, and the contents and organization of the output file is subject to change. The current organization and a brief explanation of HDF5 is provided in the [Dakota HDF5 Output](#) section of this manual.

Examples

For the example input in the parent keyword, the following interface groups would be written:

- `/interfaces/truth/truth_m/`
- `/interfaces/APPROX_INTERFACE_1/surr/`

Depending on the [model_selection](#), the following links may be added to model `sources` groups.

- `/models/simulation/truth_m/sources/truth → /interfaces/truth/truth_m/`
- `/models/surrogate/surr/sources/APPROX_INTERFACE_1 → /interfaces/APPROX_INTERFACE_1/surr/`

7.1.8 graphics

- [Keywords Area](#)
- [environment](#)
- [graphics](#)

(DEPRECATED) Display plots of variables and responses

Topics

This keyword is related to the topics:

- [dakota_output](#)

Specification

Alias: none

Argument(s): none

Default: graphics off

Description

Activate Dakota's legacy X Windows-based graphics feature. Dakota plotting and visualization capabilities are increasingly available in the Dakota graphical user interface (GUI); see the Dakota GUI User Manual in the documentation section of the Dakota website for additional details.

The `graphics` flag activates a 2D graphics window containing history plots for the variables and response functions in the study. This window is updated approximately every two seconds. Some study types such as surrogate-based optimization or local reliability specialize the use of the graphics window.

There is no dependence between the `graphics` flag and the `tabular_data` flag; they may be used independently or concurrently.

See Also

These keywords may also be of interest:

- [tabular_data](#)

7.1.9 check

- [Keywords Area](#)
- [environment](#)
- [check](#)

Invoke Dakota in input check mode

Topics

This keyword is related to the topics:

- [command_line_options](#)

Specification

Alias: none

Argument(s): none

Default: no check; proceed to run

Description

When specified, Dakota input will be parsed and the problem instantiated. Dakota will exit reporting whether any errors were found.

7.1.10 pre_run

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)

Invoke Dakota with pre-run mode active

Topics

This keyword is related to the topics:

- [command_line_options](#)

Specification

Alias: none

Argument(s): none

Default: pre-run, run, post-run all executed

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		input	Base filename for pre-run mode data input
	Optional		output	Base filename for pre-run mode data output

Description

When specified, Dakota execution will include the pre-run mode, which sets up methods and often generates parameter sets to evaluate. This mode is currently useful for parameter study, DACE, and Monte Carlo sampling methods.

Default Behavior

When no run modes are specified, Dakota will perform pre-run, run, and post-run phases.

7.1.10.1 [input](#)

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)
- [input](#)

Base filename for pre-run mode data input

Topics

This keyword is related to the topics:

- [dakota_IO](#)

Specification

Alias: none

Argument(s): STRING

Default: no pre-run specific input read

Description

(For future expansion; not currently used by any methods.) Specify a base filename from which Dakota will read any pre-run input data. This option is overridden by any command-line `-pre_run` arguments.

7.1.10.2 output

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)
- [output](#)

Base filename for pre-run mode data output

Topics

This keyword is related to the topics:

- [dakota_IO](#)

Specification

Alias: none

Argument(s): STRING

Default: no pre-run specific output written

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specify a base filename to which Dakota will write any pre-run output data (typically parameter sets to be evaluated). This option is overridden by any command-line `-pre_run` arguments.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.1.10.2.1 `custom_annotated`

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)
- [output](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.1.10.2.1.1 header

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)
- [output](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.1.10.2.1.2 eval_id

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)

- [output](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.1.10.2.1.3 interface_id

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)
- [output](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.1.10.2.2 annotated

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)
- [output](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.1.10.2.3 freeform

- [Keywords Area](#)
- [environment](#)
- [pre_run](#)
- [output](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:


```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
          0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.1.11 run

- [Keywords Area](#)
- [environment](#)
- [run](#)

Invoke Dakota with run mode active

Topics

This keyword is related to the topics:

- [command_line_options](#)

Specification

Alias: none

Argument(s): none

Default: pre-run, run, post-run all executed

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		input	Base filename for run mode data input
	Optional		output	Base filename for run mode data output

Description

When specified, Dakota execution will include the run mode, which invokes interfaces to map parameters to responses.

Default Behavior

When no run modes are specified, Dakota will perform pre-run, run, and post-run phases.

7.1.11.1 input

- [Keywords Area](#)
- [environment](#)
- [run](#)
- [input](#)

Base filename for run mode data input

Topics

This keyword is related to the topics:

- [dakota_IO](#)

Specification

Alias: none

Argument(s): STRING

Default: no run specific input read

Description

(For future expansion; not currently used by any methods.) Specify a base filename from which Dakota will read any run input data, such as parameter sets to evaluate. This option is overridden by any command-line -run arguments.

7.1.11.2 output

- [Keywords Area](#)
- [environment](#)
- [run](#)
- [output](#)

Base filename for run mode data output

Topics

This keyword is related to the topics:

- [dakota_IO](#)

Specification

Alias: none

Argument(s): STRING

Default: no run specific output written

Description

(For future expansion; not currently used by any methods.) Specify a base filename to which Dakota will write any run output data (typically parameter, response pairs). This option is overridden by any command-line -run arguments.

7.1.12 post_run

- [Keywords Area](#)
- [environment](#)
- [post_run](#)

Invoke Dakota with post-run mode active

Topics

This keyword is related to the topics:

- [command_line_options](#)

Specification

Alias: none

Argument(s): none

Default: pre-run, run, post-run all executed

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		input	Base filename for post-run mode data input
	Optional		output	Base filename for post-run mode data output

Description

When specified, Dakota execution will include the post-run mode, which analyzes parameter/response data sets and computes final results.. This mode is currently useful for parameter study, DACE, and Monte Carlo sampling methods.

Default Behavior

When no run modes are specified, Dakota will perform pre-run, run, and post-run phases.

7.1.12.1 input

- [Keywords Area](#)
- [environment](#)

- [post_run](#)
- [input](#)

Base filename for post-run mode data input

Topics

This keyword is related to the topics:

- [dakota_IO](#)

Specification

Alias: none

Argument(s): STRING

Default: no post-run specific input read

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specify a base filename from which Dakota will read any post-run input data, such as parameter/response data on which to calculate final statistics. This option is overridden by any command-line `-post_run` arguments.

Usage Tips

Dakota imports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.1.12.1.1 `custom_annotated`

- [Keywords Area](#)
- [environment](#)
- [post_run](#)
- [input](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002      0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.1.12.1.1.1 header

- [Keywords Area](#)
- [environment](#)
- [post_run](#)
- [input](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.1.12.1.1.2 eval_id

- [Keywords Area](#)
- [environment](#)
- [post_run](#)
- [input](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.1.12.1.1.3 `interface_id`

- [Keywords Area](#)
- [environment](#)
- [post_run](#)
- [input](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.1.12.1.2 `annotated`

- [Keywords Area](#)
- [environment](#)
- [post_run](#)
- [input](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.1.12.1.3 freeform

- [Keywords Area](#)
- [environment](#)
- [post_run](#)
- [input](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.1.12.2 output

- [Keywords Area](#)
- [environment](#)
- [post_run](#)
- [output](#)

Base filename for post-run mode data output

Topics

This keyword is related to the topics:

- [dakota_IO](#)

Specification

Alias: none

Argument(s): STRING

Default: no post-run specific output written

Description

(For future expansion; not currently used by any methods.) Specify a base filename to which Dakota will write any post-run output data. This option is overridden by any command-line `-post_run` arguments.

7.1.13 `top_method_pointer`

- [Keywords Area](#)
- [environment](#)
- [top_method_pointer](#)

Identify which method leads the Dakota study

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: `method_pointer`

Argument(s): STRING

Default: see discussion

Description

An optional `top_method_pointer` specification may be used to point to a particular method specification that will lead the Dakota analysis. The associated string must be a method identifier specified via `id_method`. If `top_method_pointer` is not used, then it will be inferred as described below (no `top_method_pointer` within an environment specification is treated the same as no environment specification).

Default Behavior

The `top_method_pointer` keyword is typically used in Dakota studies consisting of more than one [method](#) block to clearly indicate which is the leading method. This method provides the starting point for the iteration. The corresponding method specification may recurse with additional sub-method pointers in the case of "meta-iteration" (see [method](#)) or may specify a single method without recursion. Either case will ultimately result in identification

of one or more model specifications using `model_pointer`, which again may or may not involve further recursion (see [nested](#) and [surrogate](#) for recursion cases). Each of the model specifications identify the variables and responses specifications (using [variables_pointer](#) and [responses_pointer](#)) that are used to build the model, and depending on the type of model, may also identify an interface specification (for example, using [interface_pointer](#)). If one of these specifications does not provide an optional pointer, then that component will be constructed using the last specification parsed.

When the environment block is omitted, the top level method will be inferred as follows: When a single method is specified, there is no ambiguity and the sole method will be the top method. When multiple methods are specified, the top level method will be deduced from the hierarchical relationships implied by method pointers. If this inference is not well defined (e.g., there are multiple method specifications without any pointer relationship), then the default behavior is to employ the last method specification parsed.

Examples

Specify that the optimization method is the outermost method in an optimization under uncertainty study

```
environment
  top_method_pointer 'OPTIMIZATION_METHOD'
method
  id_method 'UQ_METHOD'
...
method
  id_method 'OPTIMIZATION_METHOD'
...
```

See Also

These keywords may also be of interest:

- [id_method](#)

7.2 method

- [Keywords Area](#)
- [method](#)

Begins Dakota method selection and behavioral settings.

Topics

This keyword is related to the topics:

- [block](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		id_method	Name the method block; helpful when there are multiple
	Optional		output	Control how much method information is written to the screen and output file
	Optional		final_solutions	Number of designs returned as the best solutions
			hybrid	Strategy in which a set of methods synergistically seek an optimal design
			multi_start	Multi-Start Optimization Method
			pareto_set	Pareto set optimization
			branch_and_bound	(Experimental Capability) Solves a mixed integer nonlinear optimization problem
			surrogate_based_-local	Local Surrogate Based Optimization
			surrogate_based_-global	Adaptive Global Surrogate-Based Optimization
			dot_frcg	DOT conjugate gradient optimization method
			dot_mmfd	DOT modified method of feasible directions
			dot_bfgs	DOT BFGS optimization method

Required (Choose One)

Method (Iterative Algorithm) (Group 1)

dot_slp	DOT Sequential Linear Program
dot_sqp	DOT Sequential Quadratic Program
conmin_frcg	CONMIN conjugate gradient optimization method
conmin_mfd	CONMIN method of feasible directions
dl_solver	(Experimental) Dynamically-loaded solver
npsol_sqp	NPSOL Sequential Quadratic Program
nlssol_sqp	Sequential Quadratic Program for nonlinear least squares
nlpql_sqp	NLPQL Sequential Quadratic Program
optpp_cg	A conjugate gradient optimization method
optpp_q_newton	Quasi-Newton optimization method
optpp_fd_newton	Finite Difference Newton optimization method
optpp_g_newton	Newton method based least-squares calibration
optpp_newton	Newton method based optimization
optpp_pds	Simplex-based derivative free optimization method

demo_tpl	Third-party optimization library integration demonstration.
rol	Rapid Optimization Library (ROL) is a large-scale optimization package within Trilinos.
asynch_pattern_search	Pattern search, derivative free optimization method
mesh_adaptive_search	Finds optimal variable values using adaptive mesh-based search
nowpac	Gradient-free inequality-constrained optimization using Nonlinear Optimization With Path Augmented Constraints (NOWPAC).
snowpac	Stochastic version of NOWPAC that incorporates error estimates and noise mitigation.
moga	Multi-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)
soga	Single-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)
coliny_pattern_search	Pattern search, derivative free optimization method

coliny_solis_wets	Simple greedy local search method
coliny_cobyla	Constrained Optimization BY Linear Approximations (COBYLA)
coliny_direct	Dividing RECTangles method
coliny_ea	Evolutionary Algorithm
coliny_beta	(Experimental) Coliny beta solver
nl2sol	Trust-region method for nonlinear least squares
nonlinear_cg	(Experimental) nonlinear conjugate gradient optimization
ncsu_direct	Dividing RECTangles method
genie_opt_darts	Voronoi-based high-dimensional global Lipschitzian optimization
genie_direct	Classical high-dimensional global Lipschitzian optimization Classical high-dimensional global Lipschitzian optimization
efficient_global	Global Surrogate Based Optimization, a.k.a. EGO
surrogate_based_uq	Generic UQ method for constructing and interrogating a surrogate model.

function_train	UQ method leveraging a functional tensor train surrogate model.
multifidelity_-function_train	Multifidelity uncertainty quantification using function train expansions
multilevel_function-_train	Multilevel uncertainty quantification using function train expansions
polynomial_chaos	Uncertainty quantification using polynomial chaos expansions
multifidelity_-polynomial_chaos	Multifidelity uncertainty quantification using polynomial chaos expansions
multilevel_-polynomial_chaos	Multilevel uncertainty quantification using polynomial chaos expansions
stoch_collocation	Uncertainty quantification with stochastic collocation
multifidelity_stoch-_collocation	Multifidelity uncertainty quantification using stochastic collocation
sampling	Randomly samples variables according to their distributions

multilevel_sampling	Multilevel sampling methods for UQ
multifidelity_sampling	Multifidelity sampling methods for UQ
multilevel_multifidelity_sampling	Multilevel--Multifidelity sampling methods for UQ
approximate_control_variate	Approximate control variate (ACV) sampling methods for UQ
importance_sampling	Importance sampling
gpais	Gaussian Process Adaptive Importance Sampling
adaptive_sampling	(Experimental) Adaptively refine a Gaussian process surrogate
pof_darts	Probability-of-Failure (POF) darts is a novel method for estimating the probability of failure based on random sphere-packing.
rkd_darts	Recursive k-d (RKD) Darts: Recursive Hyperplane Sampling for Numerical Integration of High-Dimensional Functions.

global_evidence	Evidence theory with evidence measures computed with global optimization methods
global_interval_est	Interval analysis using global optimization methods
bayes_calibration	Bayesian calibration
dace	Design and Analysis of Computer Experiments
fsu_cvt	Design of Computer Experiments - Centroidal Voronoi Tessellation
psuade_moat	Morris One-at-a-Time
local_evidence	Evidence theory with evidence measures computed with local optimization methods
local_interval_est	Interval analysis using local optimization
local_reliability	Local reliability method
global_reliability	Global reliability methods
fsu_quasi_mc	Design of Computer Experiments - Quasi-Monte Carlo sampling
vector_parameter_study	Samples variables along a user-defined vector
list_parameter_study	Samples variables as a specified values

			centered_-parameter_study	Samples variables along points moving out from a center point
			multidim_-parameter_study	Samples variables on full factorial grid of study points
			richardson_extrap	Estimate order of convergence of a response as model fidelity increases

Description

The `method` keyword signifies the start of a block in the Dakota input file. A method block contains the various keywords necessary to select a method and to control its behavior.

Method Block Requirements

At least one `method` block must appear in the Dakota input file. Multiple `method` blocks may be needed to fully define advanced analysis approaches.

Each `method` block must specify one method and, optionally, any associated keywords that govern the behavior of the method.

The Methods

Each `method` block must select one method.

Starting with Dakota v6.0, the methods are grouped into two types: standard methods and multi-component methods.

The standard methods are stand-alone and self-contained in the sense that they only require a model to perform a study. They do not call other methods. While methods such as `polynomial_chaos` and `efficient_global` internally utilize multiple iterator and surrogate model components, these components are generally hidden from user control due to restrictions on modularity; thus, these methods are stand-alone.

The multi-component group of methods provides a higher level "meta-algorithm" that points to other methods and models that support sub-iteration. For example, in a sequential hybrid method, the `hybrid` method specification must identify a list of subordinate methods, and the "meta-algorithm" executes these methods in sequence and transfers information between them. Surrogate-based minimizers provide another example in that they point both to other methods (e.g. what optimization method is used to solve the approximate subproblem) as well as to models (e.g. what type of surrogate model is employed). Multi-component methods generally provide some level of "plug and play" modularity, through their flexible support of a variety of method and model selections.

Component-Based Iterator Commands

Component-based iterator specifications include hybrid, multi-start, pareto set, surrogate-based local, surrogate-based global, and branch and bound methods. Whereas a standard iterator specification only needs an optional model pointer string (specified with `model_pointer`), component-based iterator specifications can include method pointer, method name, and model pointer specifications in order to define the components employed in the "meta-iteration." In particular, these specifications identify one or more methods (by pointer or by name) to specify the subordinate iterators that will be used in the top-level algorithm. Identifying a sub-iterator by name instead of by pointer is a lightweight option that relaxes the need for a separate method specification for the sub-iterator; however, a model pointer may be required in this case to provide the specification connectivity normally supported by the method pointer. Refer to these individual method descriptions for specific requirements for these advanced methods.

Method Independent Controls

In addition to the method, there are 10 optional keywords, which are referred to as method independent controls. These controls are valid for enough methods that it was reasonable to pull them out of the method dependent blocks

and consolidate the specifications, however, they are NOT universally respected by all methods.

Examples

Several examples follow. The first example shows a minimal specification for an optimization method.

```
method
  dot_sqp
```

This example uses all of the defaults for this method.

A more sophisticated example would be

```
method,
  id_method = 'NLP1'
  dot_sqp
  max_iterations = 50
  convergence_tolerance = 1e-4
  output verbose
  model_pointer = 'M1'
```

This example demonstrates the use of identifiers and pointers as well as some method independent and method dependent controls for the sequential quadratic programming (SQP) algorithm from the DOT library. The `max_` iterations, `convergence_tolerance`, and `output` settings are method independent controls, in that they are defined for a variety of methods.

The next example shows a specification for a least squares method.

```
method
  optpp_g_newton
  max_iterations = 10
  convergence_tolerance = 1.e-8
  search_method trust_region
  gradient_tolerance = 1.e-6
```

Some of the same method independent controls are present along with several method dependent controls (`search_method` and `gradient_tolerance`) which are only meaningful for OPT++ methods (see [package_optpp](#)).

The next example shows a specification for a nondeterministic method with several method dependent controls (refer to [sampling](#)).

```
method
  sampling
  samples = 100
  seed = 12345
  sample_type lhs
  response_levels = 1000. 500.
```

The last example shows a specification for a parameter study method where, again, each of the controls are method dependent (refer to [vector_parameter_study](#)).

```
method
  vector_parameter_study
  step_vector = 1. 1. 1.
  num_steps = 10
```

7.2.1 id_method

- [Keywords Area](#)
- [method](#)
- [id_method](#)

Name the method block; helpful when there are multiple

Topics

This keyword is related to the topics:

- [block_identifier](#)
- [method_independent_controls](#)

Specification

Alias: none

Argument(s): STRING

Default: strategy use of last method parsed

Description

The method identifier string is supplied with `id_method` and is used to provide a unique identifier string for use with environment or meta-iterator specifications (refer to [environment](#)). It is appropriate to omit a method identifier string if only one method is included in the input file, since the single method to use is unambiguous in this case.

7.2.2 output

- [Keywords Area](#)
- [method](#)
- [output](#)

Control how much method information is written to the screen and output file

Topics

This keyword is related to the topics:

- [dakota_output](#)
- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: normal

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Output Level (Group 1)	debug	Level 5 of 5 - maximum

			verbose	Level 4 of 5 - more than normal
			normal	Level 3 of 5 - default
			quiet	Level 2 of 5 - less than normal
			silent	Level 1 of 5 - minimum

Description

Choose from a total of five output levels during the course of a Dakota study. If there is no user specification for output verbosity, then the default setting is `normal`.

Specific mappings are as follows:

- `silent` (i.e., really quiet): silent iterators, silent model, silent interface, quiet approximation, quiet file operations
- `quiet`: quiet iterators, quiet model, quiet interface, quiet approximation, quiet file operations
- `normal`: normal iterators, normal model, normal interface, quiet approximation, quiet file operations
- `verbose`: verbose iterators, normal model, verbose interface, verbose approximation, verbose file operations
- `debug` (i.e., really verbose): debug iterators, normal model, debug interface, verbose approximation, verbose file operations

Note that iterators and interfaces utilize the full granularity in verbosity, whereas models, approximations, and file operations do not. With respect to iterator verbosity, different iterators implement this control in slightly different ways (as described below in the method independent controls descriptions for each iterator), however the meaning is consistent.

For models, interfaces, approximations, and file operations, `quiet` suppresses parameter and response set reporting and `silent` further suppresses function evaluation headers and scheduling output. Similarly, `verbose` adds file management, approximation evaluation, and global approximation coefficient details, and `debug` further adds diagnostics from nonblocking schedulers.

7.2.2.1 `debug`

- [Keywords Area](#)
- [method](#)
- [output](#)
- [debug](#)

Level 5 of 5 - maximum

Specification

Alias: none

Argument(s): none

Description

This is described on [output](#)

7.2.2.2 verbose

- [Keywords Area](#)
- [method](#)
- [output](#)
- [verbose](#)

Level 4 of 5 - more than normal

Specification

Alias: none

Argument(s): none

Description

This is described on [output](#)

7.2.2.3 normal

- [Keywords Area](#)
- [method](#)
- [output](#)
- [normal](#)

Level 3 of 5 - default

Specification

Alias: none

Argument(s): none

Description

This is described on [output](#)

7.2.2.4 quiet

- [Keywords Area](#)
- [method](#)
- [output](#)
- [quiet](#)

Level 2 of 5 - less than normal

Specification

Alias: none

Argument(s): none

Description

This is described on [output](#)

7.2.2.5 silent

- [Keywords Area](#)
- [method](#)
- [output](#)
- [silent](#)

Level 1 of 5 - minimum

Specification

Alias: none

Argument(s): none

Description

This is described on [output](#)

7.2.3 final_solutions

- [Keywords Area](#)
- [method](#)
- [final_solutions](#)

Number of designs returned as the best solutions

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The `final_solutions` controls the number of final solutions returned by the iterator as the best solutions.

For most optimizers, this is one, but some optimizers can produce multiple solutions (e.g. genetic algorithms).

When using a `hybrid` strategy, the number of final solutions dictates how many solutions are passed from one method to another.

Examples

In the case of sampling methods, if one specifies 100 samples (for example) but also specifies `final_solutions = 5`, the five best solutions (in order of lowest response function value) are returned.

7.2.4 hybrid

- [Keywords Area](#)
- [method](#)
- [hybrid](#)

Strategy in which a set of methods synergistically seek an optimal design

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Hybrid Method Type (Group 1)	sequential	Methods are run one at a time, in sequence
			embedded	A subordinate local method provides periodic refinements to a top-level global method
			collaborative	Multiple methods run concurrently and share information

Description

In a hybrid minimization method (`hybrid`), a set of methods synergistically seek an optimal design. The relationships among the methods are categorized as:

- collaborative
- embedded
- sequential

The goal in each case is to exploit the strengths of different optimization and nonlinear least squares algorithms at different stages of the minimization process. Global + local hybrids (e.g., genetic algorithms combined with nonlinear programming) are a common example in which the desire for identification of a global optimum is balanced with the need for efficient navigation to a local optimum.

7.2.4.1 sequential

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)

Methods are run one at a time, in sequence

Specification

Alias: uncoupled

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sub-method Selection (Group 1)	method_name_list	List of Dakota methods to sequentially or collaboratively run
			method_pointer_list	Pointers to methods to execute sequentially or collaboratively
	Optional		iterator_servers	Specify the number of iterator servers when Dakota is run in parallel
	Optional		iterator_scheduling	Specify the scheduling of concurrent iterators when Dakota is run in parallel
	Optional		processors_per_iterator	Specify the number of processors per iterator server when Dakota is run in parallel

Description

In the `sequential` approach, methods are run one at a time, in sequence. The best solutions from one method are used to initialize the next method.

The sequence of methods (i.e. iterators) to run are specified using either a `method_pointer_list` or a `method_name_list` (with optional `model_pointer_list`). Any number of iterators may be specified.

Method switching is managed through the separate convergence controls of each method. The number of solutions transferred between methods is specified by the particular method through its `final_solutions` method control.

For example, if one sets up a two-level study with a first method that generates multiple solutions such as a genetic algorithm, followed by a second method that is initialized only at a single point such as a gradient-based algorithm, it is possible to take the multiple solutions generated by the first method and create several instances of the second method, each one with a different initial starting point.

The logic governing the transfer of multiple solutions between methods is as follows:

- if one solution is returned from method A, then one solution is transferred to method B.
- If multiple solutions are returned from method A, and method B can accept multiple solutions as input (for example, as a genetic algorithm population), then one instance of method B is initialized with multiple solutions.
- If multiple solutions are returned from method A but method B only can accept one initial starting point, then method B is run several times, each one with a separate starting point from the results of method A.

7.2.4.1.1 method_name_list

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [method_name_list](#)

List of Dakota methods to sequentially or collaboratively run

Specification

Alias: none

Argument(s): STRINGLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		model_pointer_list	Associate models with method names

Description

`method_name_list` specifies a list of Dakota methods (e.g. [soga](#), [conmin_frcg](#)) that will be run by a `hybrid` `sequential` or `hybrid collaborative` method. The methods are executed with default options. The optional `model_pointer_list` may be used to associate a model with each method.

7.2.4.1.1.1 model_pointer_list

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [method_name_list](#)
- [model_pointer_list](#)

Associate models with method names

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRINGLIST

Description

Using the optional keyword `model_pointer_list`, models can be assigned to methods specified in the `method_name_list`. Models are referred to by name (i.e. by their `id_model` labels). The length of the `model_pointer_list` must be either 1 or match the length of the `method_name_list`. If the former, the same model will be used for all methods, and if the latter, methods and models will be paired in the order that they appear in the two lists.

7.2.4.1.2 `method_pointer_list`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [method_pointer_list](#)

Pointers to methods to execute sequentially or collaboratively

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRINGLIST

Description

`method_pointer_list` specifies by name the methods that are to be executed by a `hybrid sequential` or `hybrid collaborative` method. Its argument is a list of strings that refer to method blocks by name (i.e. to their `id_method` labels).

7.2.4.1.3 `iterator_servers`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [iterator_servers](#)

Specify the number of iterator servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_servers` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to `ParallelLibrary` and the `Parallel Computing` chapter of the Users Manual [5] for additional information.

7.2.4.1.4 `iterator_scheduling`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [iterator_scheduling](#)

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Scheduling Mode (Group 1)	master	Specify a dedicated master partition for parallel iterator scheduling
			peer	Specify a peer partition for parallel iterator scheduling

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_scheduling` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.4.1.4.1 master

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [iterator_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

7.2.4.1.4.2 peer

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [iterator_scheduling](#)
- [peer](#)

Specify a peer partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of `evaluation_scheduling`, it is not possible to specify `static` or `dynamic`.

7.2.4.1.5 processors_per_iterator

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [sequential](#)
- [processors_per_iterator](#)

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.4.2 embedded

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)

A subordinate local method provides periodic refinements to a top-level global method

Specification

Alias: coupled

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Global Sub-method Selection (Group 1)	global_method_ - name	Specify the global method by Dakota name
			global_method_ - pointer	Pointer to global method
	Required (<i>Choose One</i>)	Local Sub-method Selection (Group 2)	local_method_ - name	Specify the local method by Dakota name
			local_method_ - pointer	Pointer to local method
	Optional		local_search_ - probability	Probability of executing local searches
	Optional		iterator_servers	Specify the number of iterator servers when Dakota is run in parallel
	Optional		iterator_scheduling	Specify the scheduling of concurrent iterators when Dakota is run in parallel

	Optional	processors_per_iterator	Specify the number of processors per iterator server when Dakota is run in parallel
--	-----------------	---	---

Description

In the `embedded` approach, a tightly-coupled hybrid is employed in which a subordinate local method provides periodic refinements to a top-level global method.

Global and local method strings supplied with the `global_method_pointer` and `local_method_pointer` specifications identify the two methods to be used. Alternatively, Dakota method names (e.g. `'soga'`) can be supplied using the `global_method_name` and `local_method_name` keywords, which each have optional model pointer specifications. The `local_search_probability` setting is an optional specification for supplying the probability (between 0.0 and 1.0) of employing local search to improve estimates within the global search.

7.2.4.2.1 `global_method_name`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [global_method_name](#)

Specify the global method by Dakota name

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		global_model_pointer	Pointer to model used by global method

Description

`global_method_name` is used to specify the global method in a `hybrid embedded` optimization by Dakota name (e.g. `'soga'`). The name of the method is provided as a string. The method is executed with default options.

7.2.4.2.1.1 `global_model_pointer`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)

- [global_method_name](#)
- [global_model_pointer](#)

Pointer to model used by global method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Description

`global_model_pointer` can be used to specify a model for use with the Dakota method named by the `global_method_name` specification. The argument is a string that refers to the `id_model` label of the desired model.

7.2.4.2.2 `global_method_pointer`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [global_method_pointer](#)

Pointer to global method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Description

The `global_method_pointer` identifies the method block to use as the global method in a hybrid embedded optimization using its `id_method` label.

7.2.4.2.3 local_method_name

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [local_method_name](#)

Specify the local method by Dakota name

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		local_model_pointer	Pointer to model used by local method

Description

`local_method_name` is used to specify the local method in a `hybrid` `embedded` optimization by Dakota name (e.g. `'conmin_mfd'`). The name of the method is provided as a string. The method is executed with default options.

7.2.4.2.3.1 local_model_pointer

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [local_method_name](#)
- [local_model_pointer](#)

Pointer to model used by local method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Description

`local_model_pointer` can be used to specify a model for use with the Dakota method named by the `local_method_name` specification. The argument is a string that refers to the `id_model` label of the desired model.

7.2.4.2.4 local_method_pointer

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [local_method_pointer](#)

Pointer to local method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Description

The `local_method_pointer` identifies the method block to use as the local method in a `hybrid` embedded optimization using its `id_method`

7.2.4.2.5 local_search_probability

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [local_search_probability](#)

Probability of executing local searches

Specification

Alias: none

Argument(s): REAL

Description

The `local_search_probability` setting is an optional specification for supplying the probability (between 0.0 and 1.0) of employing local search to improve estimates within the global search. Its default value is 0.1.

7.2.4.2.6 `iterator_servers`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [iterator_servers](#)

Specify the number of iterator servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_servers` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.4.2.7 `iterator_scheduling`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [iterator_scheduling](#)

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group Scheduling Mode (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		master	Specify a dedicated master partition for parallel iterator scheduling
			peer	Specify a peer partition for parallel iterator scheduling

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_scheduling` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.4.2.7.1 master

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [iterator_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

7.2.4.2.7.2 peer

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [iterator_scheduling](#)
- [peer](#)

Specify a peer partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of `evaluation_scheduling`, it is not possible to specify `static` or `dynamic`.

7.2.4.2.8 processors_per_iterator

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [embedded](#)
- [processors_per_iterator](#)

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.4.3 collaborative

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)

Multiple methods run concurrently and share information

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group Sub-method Selection (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		method_name_list	List of Dakota methods to sequentially or collaboratively run
			method_pointer_list	Pointers to methods to execute sequentially or collaboratively
	Optional		iterator_servers	Specify the number of iterator servers when Dakota is run in parallel
	Optional		iterator_scheduling	Specify the scheduling of concurrent iterators when Dakota is run in parallel
	Optional		processors_per_iterator	Specify the number of processors per iterator server when Dakota is run in parallel

Description

In the collaborative approach, multiple methods work together and share solutions while executing concurrently. A list of method strings specifies the pool of iterators to be used. Any number of iterators may be specified. The

method collaboration logic follows that of either the Agent-Based Optimization or HOPSPACK codes and is currently under development and not available at this time.

7.2.4.3.1 method_name_list

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [method_name_list](#)

List of Dakota methods to sequentially or collaboratively run

Specification

Alias: none

Argument(s): STRINGLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		model_pointer_list	Associate models with method names

Description

`method_name_list` specifies a list of Dakota methods (e.g. [soga](#), [conmin_frcg](#)) that will be run by a `hybrid` sequential or `hybrid collaborative` method. The methods are executed with default options. The optional `model_pointer_list` may be used to associate a model with each method.

7.2.4.3.1.1 model_pointer_list

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [method_name_list](#)
- [model_pointer_list](#)

Associate models with method names

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRINGLIST

Description

Using the optional keyword `model_pointer_list`, models can be assigned to methods specified in the `method_name_list`. Models are referred to by name (i.e. by their `id_model` labels). The length of the `model_pointer_list` must be either 1 or match the length of the `method_name_list`. If the former, the same model will be used for all methods, and if the latter, methods and models will be paired in the order that they appear in the two lists.

7.2.4.3.2 `method_pointer_list`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [method_pointer_list](#)

Pointers to methods to execute sequentially or collaboratively

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRINGLIST

Description

`method_pointer_list` specifies by name the methods that are to be executed by a `hybrid sequential` or `hybrid collaborative` method. Its argument is a list of strings that refer to method blocks by name (i.e. to their `id_method` labels).

7.2.4.3.3 `iterator_servers`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [iterator_servers](#)

Specify the number of iterator servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_servers` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.4.3.4 `iterator_scheduling`

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [iterator_scheduling](#)

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Scheduling Mode (Group 1)	master	Specify a dedicated master partition for parallel iterator scheduling

			peer	Specify a peer partition for parallel iterator scheduling
--	--	--	----------------------	---

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_scheduling` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.4.3.4.1 master

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [iterator_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

7.2.4.3.4.2 peer

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [iterator_scheduling](#)

- [peer](#)

Specify a peer partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of `evaluation_scheduling`, it is not possible to specify `static` or `dynamic`.

7.2.4.3.5 processors_per_iterator

- [Keywords Area](#)
- [method](#)
- [hybrid](#)
- [collaborative](#)
- [processors_per_iterator](#)

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.5 multi_start

- [Keywords Area](#)
- [method](#)
- [multi_start](#)

Multi-Start Optimization Method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sub-method Selection (Group 1)	method_name	Specify sub-method by name
			method_pointer	Pointer to sub-method to run from each starting point
	Optional		random_starts	Number of random starting points
	Optional		starting_points	List of user-specified starting points
	Optional		iterator_servers	Specify the number of iterator servers when Dakota is run in parallel
	Optional		iterator_scheduling	Specify the scheduling of concurrent iterators when Dakota is run in parallel
	Optional		processors_per_iterator	Specify the number of processors per iterator server when Dakota is run in parallel

Description

In the multi-start iteration method (`multi_start`), a series of iterator runs are performed for different values of parameters in the model. A common use is for multi-start optimization (i.e., different local optimization runs from different starting points for the design variables), but the concept and the code are more general. Multi-start iteration is implemented within the Metalterator branch of the Iterator hierarchy within the ConcurrentMetalterator class. Additional information on the multi-start algorithm is available in the Users Manual [5].

The `multi_start` meta-iterator must specify a sub-iterator using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This iterator is responsible for completing a series of iterative analyses from a set of different starting points. These starting points can be specified as follows: (1) using `random_starts`, for which the specified number of starting points are selected randomly within the variable bounds, (2)

using `starting_points`, in which the starting values are provided in a list, or (3) using both `random_starts` and `starting_points`, for which the combined set of points will be used. In aggregate, at least one starting point must be specified. The most common example of a multi-start algorithm is multi-start optimization, in which a series of optimizations are performed from different starting values for the design variables. This can be an effective approach for problems with multiple minima.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the starting points for each sub-iterator it runs, as well as the best parameters and responses returned by each sub-iterator. See the [Multistart and Pareto Set](#) documentation for details.

7.2.5.1 `method_name`

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [method_name](#)

Specify sub-method by name

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		model_pointer	Identifier for model block to be used by a method

Description

The `method_name` keyword is used to specify a sub-method by Dakota method name (e.g. 'npsol_sqp') rather than block pointer. The method will be executed using its default settings. The optional `model_pointer` specification can be used to associate a model block with the method.

7.2.5.1.1 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [method_name](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```
variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.5.2 method_pointer

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [method_pointer](#)

Pointer to sub-method to run from each starting point

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Description

The `method_pointer` keyword is used to specify a pointer to the sub-method block that will be run from each starting point.

7.2.5.3 random_starts

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [random_starts](#)

Number of random starting points

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed	Seed of the random number generator

Description

The `multi_start` meta-iterator must specify a sub-iterator using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This iterator is responsible for completing a series of iterative analyses from a set of different starting points. These starting points can be specified as follows: (1) using `random_starts`, for which the specified number of starting points are selected randomly within the variable bounds, (2) using `starting_points`, in which the starting values are provided in a list, or (3) using both `random_starts` and `starting_points`, for which the combined set of points will be used.

7.2.5.3.1 `seed`

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [random_starts](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.5.4 starting_points

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [starting_points](#)

List of user-specified starting points

Specification

Alias: none

Argument(s): REALLIST

Description

The `multi_start` meta-iterator must specify a sub-iterator using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This iterator is responsible for completing a series of iterative analyses from a set of different starting points. These starting points can be specified as follows: (1) using `random_starts`, for which the specified number of starting points are selected randomly within the variable bounds, (2) using `starting_points`, in which the starting values are provided in a list, or (3) using both `random_starts` and `starting_points`, for which the combined set of points will be used.

7.2.5.5 iterator_servers

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [iterator_servers](#)

Specify the number of iterator servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_servers` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.5.6 `iterator_scheduling`

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [iterator_scheduling](#)

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Scheduling Mode (Group 1)	Dakota Keyword <code>master</code>	Dakota Keyword Description Specify a dedicated master partition for parallel iterator scheduling
			<code>peer</code>	Specify a peer partition for parallel iterator scheduling

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_scheduling` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.5.6.1 master

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [iterator_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

7.2.5.6.2 peer

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [iterator_scheduling](#)
- [peer](#)

Specify a peer partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of `evaluation_scheduling`, it is not possible to specify `static` or `dynamic`.

7.2.5.7 processors_per_iterator

- [Keywords Area](#)
- [method](#)
- [multi_start](#)
- [processors_per_iterator](#)

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to `ParallelLibrary` and the `Parallel Computing` chapter of the Users Manual [5] for additional information.

7.2.6 pareto_set

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)

Pareto set optimization

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sub-method Selection (Group 1)	method_name	Specify sub-method by name
			method_pointer	Pointer to optimization or least-squares sub-method
	Optional		random_weight_sets	Number of random weighting sets
	Optional		weight_sets	List of user-specified weighting sets
	Optional		iterator_servers	Specify the number of iterator servers when Dakota is run in parallel
	Optional		iterator_scheduling	Specify the scheduling of concurrent iterators when Dakota is run in parallel
	Optional		processors_per_iterator	Specify the number of processors per iterator server when Dakota is run in parallel

Description

In the pareto set minimization method (`pareto_set`), a series of optimization or least squares calibration runs are performed for different weightings applied to multiple objective functions. This set of optimal solutions defines a "Pareto set," which is useful for investigating design trade-offs between competing objectives. The code is similar enough to the `multi_start` technique that both algorithms are implemented in the same `ConcurrentMetalterator` class.

The `pareto_set` specification must identify an optimization or least squares calibration method using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This minimizer is responsible for computing a set of optimal solutions from a set of response weightings (multi-objective weights or least squares term weights). These weightings can be specified as follows: (1) using `random_weight_sets`, in which case weightings are selected randomly within [0,1] bounds, (2) using `weight_sets`, in which the weighting sets are specified in a list, or (3) using both `random_weight_sets` and `weight_sets`, for which the combined set of weights will be used. In aggregate, at least one set of weights must be specified. The set of optimal solutions is called the "pareto set," which can provide valuable design trade-off information when there are competing objectives.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the best parameters and responses returned by each sub-iterator. The weights are provided as metadata. See the [Multistart and Pareto Set](#) documentation for details.

7.2.6.1 `method_name`

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)

- [method_name](#)

Specify sub-method by name

Specification

Alias: `opt_method_name`

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		model_pointer	Identifier for model block to be used by a method

Description

The `method_name` keyword is used to specify a sub-method by Dakota method name (e.g. 'npsol_sqp') rather than block pointer. The method will be executed using its default settings. The optional `model_pointer` specification can be used to associate a model block with the method.

7.2.6.1.1 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [method_name](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: `opt_model_pointer`

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.6.2 method_pointer

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)

- [method_pointer](#)

Pointer to optimization or least-squares sub-method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: opt_method_pointer

Argument(s): STRING

Description

The `method_pointer` keyword is used to specify a pointer to an optimization or least-squares sub-method that is responsible for computing a set of optimal solutions for a set of response weightings.

7.2.6.3 random_weight_sets

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [random_weight_sets](#)

Number of random weighting sets

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed	Seed of the random number generator

Description

The `pareto_set` specification must identify an optimization or least squares calibration method using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This minimizer is responsible for computing a set of optimal solutions from a set of response weightings (multi-objective weights or least squares term weights). These weightings can be specified as follows: (1) using `random_weight_sets`, in which case weightings are selected randomly within [0,1] bounds, (2) using `weight_sets`, in which the weighting sets are specified in a list, or (3) using both `random_weight_sets` and `weight_sets`, for which the combined set of weights will be used. In aggregate, at least one set of weights must be specified. The set of optimal solutions is called the "pareto set," which can provide valuable design trade-off information when there are competing objectives.

7.2.6.3.1 seed

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [random_weight_sets](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.6.4 weight_sets

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [weight_sets](#)

List of user-specified weighting sets

Specification

Alias: multi_objective_weight_sets

Argument(s): REALLIST

Description

The `pareto_set` specification must identify an optimization or least squares calibration method using either a `method_pointer` or a `method_name` plus optional `model_pointer`. This minimizer is responsible for computing a set of optimal solutions from a set of response weightings (multi-objective weights or least squares term weights). These weightings can be specified as follows: (1) using `random_weight_sets`, in which case weightings are selected randomly within [0,1] bounds, (2) using `weight_sets`, in which the weighting sets are specified in a list, or (3) using both `random_weight_sets` and `weight_sets`, for which the combined set of weights will be used. In aggregate, at least one set of weights must be specified. The set of optimal solutions is called the "pareto set," which can provide valuable design trade-off information when there are competing objectives.

7.2.6.5 iterator_servers

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [iterator_servers](#)

Specify the number of iterator servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_servers` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to `ParallelLibrary` and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.6.6 iterator_scheduling

- [Keywords Area](#)
- [method](#)

- [pareto_set](#)
- [iterator_scheduling](#)

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Scheduling Mode (Group 1)	Dakota Keyword master	Dakota Keyword Description Specify a dedicated master partition for parallel iterator scheduling
			peer	Specify a peer partition for parallel iterator scheduling

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_scheduling` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.6.6.1 master

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [iterator_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

7.2.6.6.2 peer

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [iterator_scheduling](#)
- [peer](#)

Specify a peer partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of `evaluation_scheduling`, it is not possible to specify `static` or `dynamic`.

7.2.6.7 processors_per_iterator

- [Keywords Area](#)
- [method](#)
- [pareto_set](#)
- [processors_per_iterator](#)

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.2.7 `branch_and_bound`

- [Keywords Area](#)
- [method](#)
- [branch_and_bound](#)

(Experimental Capability) Solves a mixed integer nonlinear optimization problem

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Local Optimizer Selection (Group 1)	method_pointer	Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem

		<code>method_name</code>	Specify sub-method by name
	Optional	<code>scaling</code>	Turn on scaling for variables, responses, and constraints

Description

The branch-and-bound optimization methods solves mixed integer nonlinear optimization problems. It does so by partitioning the parameter space according to some criteria along the integer or discrete variables. It then relaxes (i.e., treats all variables as continuous) the sub-problems created by the partitions and solves each sub-problem with a continuous nonlinear optimization method. Results of the sub-problems are combined in such a way that yields the solution to the original optimization problem.

Default Behavior

Branch-and-bound expects all discrete variables to be relaxable. If your problem has categorical or otherwise non-relaxable discrete variables, then this is not the optimization method you are looking for.

Expected Output

The optimal solution and associated parameters will be printed to the screen output.

Usage Tips

The user must choose a nonlinear optimization method to solve the sub- problems. We recommend choosing a method that would be chosen to solve a continuous problem that has similar form to the mixed integer problem.

Examples

```
environment
  method_pointer = 'BandB'

method
  id_method = 'BandB'
  branch_and_bound
  output verbose
  method_pointer = 'SubNLP'

method
  id_method = 'SubNLP'
  coliny_ea
  seed = 12345
  max_iterations = 100
  max_function_evaluations = 100

variables,
  continuous_design = 3
  initial_point      -1.0   1.5   2.0
  upper_bounds       10.0  10.0  10.0
  lower_bounds       -10.0 -10.0 -10.0
  descriptors        'x1'  'x2'  'x3'
  discrete_design_range = 2
  initial_point      2     2
  lower_bounds        1     1
  upper_bounds        4     9
  descriptors        'y1'  'y2'

interface,
  fork
  analysis_driver = 'text_book'

responses,
  objective_functions = 1
```

```
nonlinear_inequality_constraints = 2
numerical_gradients
no_hessians
```

7.2.7.1 `method_pointer`

- [Keywords Area](#)
- [method](#)
- [branch_and_bound](#)
- [method_pointer](#)

Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Description

The `method_pointer` keyword is used to specify a pointer to an optimization or least-squares sub-method to apply in the context of what could be described as hierarchical methods. In surrogate-based methods, the sub-method is applied to the surrogate model. In the branch-and-bound method, the sub-method is applied to the relaxed sub-problems.

Any `model_pointer` identified in the sub-method specification is ignored. Instead, the parent method is responsible for selecting the appropriate model to use as specified by its `model_pointer`. In surrogate-based methods, it is a surrogate model defined using its `model_pointer`. In branch-and-bound methods, it is the relaxed model that is constructed internally from the original model.

7.2.7.2 `method_name`

- [Keywords Area](#)
- [method](#)
- [branch_and_bound](#)
- [method_name](#)

Specify sub-method by name

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		model_pointer	Identifier for model block to be used by a method

Description

The `method_name` keyword is used to specify a sub-method by Dakota method name (e.g. 'npsol_sqp') rather than block pointer. The method will be executed using its default settings. The optional `model_pointer` specification can be used to associate a model block with the method.

7.2.7.2.1 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [branch_and_bound](#)
- [method_name](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.7.3 scaling

- [Keywords Area](#)
- [method](#)
- [branch_and_bound](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.8 surrogate_based_local

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)

Local Surrogate Based Optimization

Topics

This keyword is related to the topics:

- [surrogate_based_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Subproblem Optimizer Selection (Group 1)	method_pointer	Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
			method_name	Specify sub-method by name
	Required		model_pointer	Identifier for model block to be used by a method
	Optional		soft_convergence_limit	Limit number of iterations w/ little improvement
	Optional		truth_surrogate_bypass	Bypass lower level surrogates when performing truth verifications on a top level surrogate
	Optional		approx_subproblem	Identify functions to be included in surrogate merit function
	Optional		merit_function	Select type of penalty or merit function
	Optional		acceptance_logic	Set criteria for trusted surrogate
	Optional		constraint_relax	Enable constraint relaxation
	Optional		trust_region	Specification group for trust region model management
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods

	Optional	convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	constraint_ - tolerance	Maximum allowable constraint violation still considered feasible

Description

In surrogate-based optimization (SBO) and surrogate-based nonlinear least squares (SBNLS), minimization occurs using a set of one or more approximations, defined from a surrogate model, that are built and periodically updated using data from a "truth" model. The surrogate model can be a global data fit (e.g., regression or interpolation of data generated from a design of computer experiments), a multipoint approximation, a local Taylor Series expansion, or a model hierarchy approximation (e.g., a low-fidelity simulation model), whereas the truth model involves a high-fidelity simulation model. The goals of surrogate-based methods are to reduce the total number of truth model simulations and, in the case of global data fit surrogates, to smooth noisy data with an easily navigated analytic function.

In the surrogate-based local method, a trust region approach is used to manage the minimization process to maintain acceptable accuracy between the surrogate model and the truth model (by limiting the range over which the surrogate model is trusted). The process involves a sequence of minimizations performed on the surrogate model and bounded by the trust region. At the end of each approximate minimization, the candidate optimum point is validated using the truth model. If sufficient decrease has been obtained in the truth model, the trust region is re-centered around the candidate optimum point and the trust region will either shrink, expand, or remain the same size depending on the accuracy with which the surrogate model predicted the truth model decrease. If sufficient decrease has not been attained, the trust region center is not updated and the entire trust region shrinks by a user-specified factor. The cycle then repeats with the construction of a new surrogate model, a minimization, and another test for sufficient decrease in the truth model. This cycle continues until convergence is attained.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when [calibration_terms](#) are specified)

Theory

For [surrogate_based_local](#) problems with nonlinear constraints, a number of algorithm formulations exist as described in [23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual [5].

See Also

These keywords may also be of interest:

- [efficient_global](#)
- [surrogate_based_global](#)

7.2.8.1 method_pointer

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [method_pointer](#)

Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: approx_method_pointer

Argument(s): STRING

Description

The `method_pointer` keyword is used to specify a pointer to an optimization or least-squares sub-method to apply in the context of what could be described as hierarchical methods. In surrogate-based methods, the sub-method is applied to the surrogate model. In the branch-and-bound method, the sub-method is applied to the relaxed sub-problems.

Any `model_pointer` identified in the sub-method specification is ignored. Instead, the parent method is responsible for selecting the appropriate model to use as specified by its `model_pointer`. In surrogate-based methods, it is a surrogate model defined using its `model_pointer`. In branch-and-bound methods, it is the relaxed model that is constructed internally from the original model.

7.2.8.2 method_name

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [method_name](#)

Specify sub-method by name

Specification

Alias: approx_method_name

Argument(s): STRING

Description

The `method_name` keyword is used to specify a sub-method by Dakota method name (e.g. 'npsol_sqp') rather than block pointer. The method will be executed using its default settings. The optional `model_pointer` specification can be used to associate a model block with the method.

7.2.8.3 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: `approx_model_pointer`

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'
```

```
method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
```

```

seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
                 0.1 0.2 0.6
                 0.1 0.2 0.6

sample_type lhs
distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.8.4 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [soft_convergence_limit](#)

Limit number of iterations w/ little improvement

Specification

Alias: none

Argument(s): INTEGER

Default: 5

Description

`soft_convergence_limit` (a soft convergence control for the `surrogate_based_local` iterations which limits the number of consecutive iterations with improvement less than the convergence tolerance). `soft_convergence`>1

counts an iteration in which a design point was rejected as an iteration with no improvement. Setting `soft_convergence=1` ignores iterations involving rejected points, instead comparing only consecutive iterations of accepted points.

7.2.8.5 `truth_surrogate_bypass`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [truth_surrogate_bypass](#)

Bypass lower level surrogates when performing truth verifications on a top level surrogate

Specification

Alias: none

Argument(s): none

Default: no bypass

Description

`truth_surrogate_bypass` (a flag for bypassing all lower level surrogates when performing truth verifications on a top level surrogate).

7.2.8.6 `approx_subproblem`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)

Identify functions to be included in surrogate merit function

Specification

Alias: none

Argument(s): none

Default: `original_primary` `original_constraints`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Objective Formulation (Group 1)	<code>original_primary</code>	Construct approximations of all primary functions

			single_objective	Construct approximation a single objective functions only
			augmented_lagrangian_objective	Augmented Lagrangian approximate subproblem formulation
			lagrangian_objective	Lagrangian approximate subproblem formulation
	Required (<i>Choose One</i>)	Constraint Formulation (Group 2)	original_constraints	Use the constraints directly
			linearized_constraints	Use linearized approximations to the constraints
			no_constraints	Don't use constraints

Description

First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (`original_primary`), a single objective function (`single_objective`) formed from the primary function surrogates, or either an augmented Lagrangian merit function (`augmented_lagrangian_objective`) or a Lagrangian merit function (`lagrangian_objective`) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition. Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`).

7.2.8.6.1 `original_primary`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)
- [original_primary](#)

Construct approximations of all primary functions

Specification

Alias: none

Argument(s): none

Description

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in [23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual [5]. First, the "primary"

functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (`original_primary`), a single objective function (`single_objective`) formed from the primary function surrogates, or either an augmented Lagrangian merit function (`augmented_lagrangian_objective`) or a Lagrangian merit function (`lagrangian_objective`) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.

7.2.8.6.2 `single_objective`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)
- [single_objective](#)

Construct approximation a single objective functions only

Specification

Alias: none

Argument(s): none

Description

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in [23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual [5]. First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (`original_primary`), a single objective function (`single_objective`) formed from the primary function surrogates, or either an augmented Lagrangian merit function (`augmented_lagrangian_objective`) or a Lagrangian merit function (`lagrangian_objective`) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.

7.2.8.6.3 `augmented_lagrangian_objective`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)
- [augmented_lagrangian_objective](#)

Augmented Lagrangian approximate subproblem formulation

Specification

Alias: none

Argument(s): none

Description

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in [23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual [5]. First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (`original_primary`), a single objective function (`single_objective`) formed from the primary function surrogates, or either an augmented Lagrangian merit function (`augmented_lagrangian_objective`) or a Lagrangian merit function (`lagrangian_objective`) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.

7.2.8.6.4 `lagrangian_objective`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)
- [lagrangian_objective](#)

Lagrangian approximate subproblem formulation

Specification

Alias: none

Argument(s): none

Description

For SBL problems with nonlinear constraints, a number of algorithm formulations exist as described in [23] and as summarized in the Advanced Examples section of the Models chapter of the Users Manual [5]. First, the "primary" functions (that is, the objective functions or calibration terms) in the approximate subproblem can be selected to be surrogates of the original primary functions (`original_primary`), a single objective function (`single_objective`) formed from the primary function surrogates, or either an augmented Lagrangian merit function (`augmented_lagrangian_objective`) or a Lagrangian merit function (`lagrangian_objective`) formed from the primary and secondary function surrogates. The former option may imply the use of a nonlinear least squares method, a multiobjective optimization method, or a single objective optimization method to solve the approximate subproblem, depending on the definition of the primary functions. The latter three options all imply the use of a single objective optimization method regardless of primary function definition.

7.2.8.6.5 `original_constraints`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)
- [original_constraints](#)

Use the constraints directly

Specification

Alias: none

Argument(s): none

Description

The surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`).

7.2.8.6.6 `linearized_constraints`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)
- [linearized_constraints](#)

Use linearized approximations to the constraints

Specification

Alias: none

Argument(s): none

Description

The surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`).

7.2.8.6.7 `no_constraints`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [approx_subproblem](#)
- [no_constraints](#)

Don't use constraints

Specification

Alias: none

Argument(s): none

Description

The surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`).

7.2.8.7 merit_function

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [merit_function](#)

Select type of penalty or merit function

Specification

Alias: none

Argument(s): none

Default: `augmented_lagrangian_merit`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Merit Function (Group 1)	penalty_merit	Use penalty merit function
			adaptive_penalty_merit	Use adaptive penalty merit function
			lagrangian_merit	Use first-order Lagrangian merit function
			augmented_lagrangian_merit	Use combined penalty and zeroth-order Lagrangian merit function

Description

Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by `surrogate_based_local` iteration number (`penalty_merit`), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (`adaptive_penalty_merit`), a Lagrangian merit function which employs first-order Lagrange multiplier updates (`lagrangian_merit`), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (`augmented_lagrangian_merit`). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [15].

7.2.8.7.1 penalty_merit

- [Keywords Area](#)

- [method](#)
- [surrogate_based_local](#)
- [merit_function](#)
- [penalty_merit](#)

Use penalty merit function

Specification

Alias: none

Argument(s): none

Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by SBL iteration number (`penalty_merit`), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (`adaptive_penalty_merit`), a Lagrangian merit function which employs first-order Lagrange multiplier updates (`lagrangian_merit`), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (`augmented_lagrangian_merit`). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [15].

7.2.8.7.2 adaptive_penalty_merit

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [merit_function](#)
- [adaptive_penalty_merit](#)

Use adaptive penalty merit function

Specification

Alias: none

Argument(s): none

Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by SBL iteration number (`penalty_merit`),

an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (`adaptive_penalty_merit`), a Lagrangian merit function which employs first-order Lagrange multiplier updates (`lagrangian_merit`), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (`augmented_lagrangian_merit`). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [15].

7.2.8.7.3 `lagrangian_merit`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [merit_function](#)
- [lagrangian_merit](#)

Use first-order Lagrangian merit function

Specification

Alias: none

Argument(s): none

Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by SBL iteration number (`penalty_merit`), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (`adaptive_penalty_merit`), a Lagrangian merit function which employs first-order Lagrange multiplier updates (`lagrangian_merit`), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (`augmented_lagrangian_merit`). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [15].

7.2.8.7.4 `augmented_lagrangian_merit`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [merit_function](#)
- [augmented_lagrangian_merit](#)

Use combined penalty and zeroth-order Lagrangian merit function

Specification

Alias: none

Argument(s): none

Description

Second, the surrogate constraints in the approximate subproblem can be selected to be surrogates of the original constraints (`original_constraints`) or linearized approximations to the surrogate constraints (`linearized_constraints`), or constraints can be omitted from the subproblem (`no_constraints`). Following optimization of the approximate subproblem, the candidate iterate is evaluated using a merit function, which can be selected to be a simple penalty function with penalty ramped by SBL iteration number (`penalty_merit`), an adaptive penalty function where the penalty ramping may be accelerated in order to avoid rejecting good iterates which decrease the constraint violation (`adaptive_penalty_merit`), a Lagrangian merit function which employs first-order Lagrange multiplier updates (`lagrangian_merit`), or an augmented Lagrangian merit function which employs both a penalty parameter and zeroth-order Lagrange multiplier updates (`augmented_lagrangian_merit`). When an augmented Lagrangian is selected for either the subproblem objective or the merit function (or both), updating of penalties and multipliers follows the approach described in [15].

7.2.8.8 acceptance_logic

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [acceptance_logic](#)

Set criteria for trusted surrogate

Specification

Alias: none

Argument(s): none

Default: filter

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Acceptance Logic (Group 1)	tr_ratio	Surrogate-Based Local iterate acceptance logic
			filter	Surrogate-Based Local iterate acceptance logic

Description

Following calculation of the merit function for the new iterate, the iterate is accepted or rejected and the trust region size is adjusted for the next `surrogate_based_local` iteration. Iterate acceptance is governed either by a trust region ratio (`tr_ratio`) formed from the merit function values or by a filter method (`filter`); however, trust region resizing logic is currently based only on the trust region ratio. For infeasible iterates, constraint relaxation can be used for balancing constraint satisfaction and progress made toward an optimum.

7.2.8.8.1 tr_ratio

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)

- [acceptance_logic](#)
- [tr_ratio](#)

Surrogate-Based Local iterate acceptance logic

Specification

Alias: none

Argument(s): none

Description

Following calculation of the merit function for the new iterate, the iterate is accepted or rejected and the trust region size is adjusted for the next SBL iteration. Iterate acceptance is governed either by a trust region ratio (`tr_ratio`) formed from the merit function values or by a filter method (`filter`); however, trust region resizing logic is currently based only on the trust region ratio. For infeasible iterates, constraint relaxation can be used for balancing constraint satisfaction and progress made toward an optimum. The command `constraint_relax` followed by a method name specifies the type of relaxation to be used. Currently, `homotopy` [71] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

7.2.8.8.2 filter

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [acceptance_logic](#)
- [filter](#)

Surrogate-Based Local iterate acceptance logic

Specification

Alias: none

Argument(s): none

Description

Following calculation of the merit function for the new iterate, the iterate is accepted or rejected and the trust region size is adjusted for the next SBL iteration. Iterate acceptance is governed either by a trust region ratio (`tr_ratio`) formed from the merit function values or by a filter method (`filter`); however, trust region resizing logic is currently based only on the trust region ratio. For infeasible iterates, constraint relaxation can be used for balancing constraint satisfaction and progress made toward an optimum. The command `constraint_relax` followed by a method name specifies the type of relaxation to be used. Currently, `homotopy` [71] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

7.2.8.9 `constraint_relax`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [constraint_relax](#)

Enable constraint relaxation

Specification

Alias: none

Argument(s): none

Default: no relaxation

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		homotopy	Surrogate-Based local constraint relaxation method for infeasible iterates

Description

The command `constraint_relax` followed by a method name specifies the type of relaxation to be used. Currently, `homotopy` [71] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

7.2.8.9.1 `homotopy`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [constraint_relax](#)
- [homotopy](#)

Surrogate-Based local constraint relaxation method for infeasible iterates

Specification

Alias: none

Argument(s): none

Description

Currently, `homotopy` [71] is the only available method for constraint relaxation, and this method is dependent on the presence of the NPSOL library within the Dakota executable.

7.2.8.10 trust_region

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [trust_region](#)

Specification group for trust region model management

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_size	Trust region initial size (relative to bounds)
	Optional		minimum_size	Trust region minimum size
	Optional		contract_threshold	Shrink trust region if trust region ratio is below this value
	Optional		expand_threshold	Expand trust region if trust region ratio is above this value
	Optional		contraction_factor	Amount by which step length is rescaled
	Optional		expansion_factor	Trust region expansion factor

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next `surrogate_based_local` iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next `surrogate_based_local` iteration.

Each of these specifications are REAL scalars, with the exception of `initial_size`, which is a REALLIST.

7.2.8.10.1 initial_size

- [Keywords Area](#)

- [method](#)
- [surrogate_based_local](#)
- [trust_region](#)
- [initial_size](#)

Trust region initial size (relative to bounds)

Specification

Alias: none

Argument(s): REALLIST

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

All of these specifications are `REAL` scalars, with the exception of the specification of `initial_size`, which is a `REALLIST`. This array corresponds to the case when there are more than 2 model forms or discretizations within a model hierarchy. The default `initial_size` involves a recursive halving of the global bounds for each trust region in the hierarchy: e.g., a scalar value of `.5` for a single trust region managing two model forms/discretizations, or an array of `(.125, .25, .5)` for four model forms/discretizations (three trust regions ordered from the lowest to highest fidelity surrogate, with model four as truth).

7.2.8.10.2 `minimum_size`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [trust_region](#)
- [minimum_size](#)

Trust region minimum size

Specification

Alias: none

Argument(s): REAL

Default: 1.e-6

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.8.10.3 `contract_threshold`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [trust_region](#)
- [contract_threshold](#)

Shrink trust region if trust region ratio is below this value

Specification

Alias: none

Argument(s): REAL

Default: 0.25

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.8.10.4 `expand_threshold`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [trust_region](#)
- [expand_threshold](#)

Expand trust region if trust region ratio is above this value

Specification

Alias: none

Argument(s): REAL

Default: 0.75

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.8.10.5 `contraction_factor`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [trust_region](#)
- [contraction_factor](#)

Amount by which step length is rescaled

Specification

Alias: none

Argument(s): REAL

Default: 0.25

Description

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction_factor}$

7.2.8.10.6 `expansion_factor`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)

- [trust_region](#)
- [expansion_factor](#)

Trust region expansion factor

Specification

Alias: none

Argument(s): REAL

Default: 2.0

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.8.11 max_iterations

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.8.12 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.8.13 constraint_tolerance

- [Keywords Area](#)
- [method](#)
- [surrogate_based_local](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.9 surrogate_based_global

- [Keywords Area](#)
- [method](#)
- [surrogate_based_global](#)

Adaptive Global Surrogate-Based Optimization

Topics

This keyword is related to the topics:

- [surrogate_based_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group Sub-method Selection (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		method_pointer	Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem
			method_name	Specify sub-method by name
	Required		model_pointer	Identifier for model block to be used by a method
	Optional		replace_points	(Recommended) Replace points in the surrogate training set, instead of appending
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods

Description

The `surrogate_based_global` method iteratively performs optimization on a global surrogate using the same bounds during each iteration.

- In one iteration, optimal solutions are found on the surrogate model, and a subset of these are passed to the next iteration.
- At the next iteration, these surrogate-optimal variable sets are evaluated with the "truth" model, and added to the set of points over which the next surrogate is constructed.

In this way, the optimization operates on a more accurate surrogate during each iteration, presumably driving to optimality quickly. In contrast to [surrogate_based_local](#), this approach has no guarantee of convergence.

Usage Tips

Attention: This adaptive method is not recommended for "build-once" surrogates trained from (static) imported data or trained online using a single Dakota design of experiments. Instead, any Dakota optimization method can be used with a (build-once) global surrogate by specifying the `id_model` of a global surrogate model with the optimizer's `model_pointer` keyword.

Configuring the method:

- The sub-method, specified with either `method_pointer` or `method_name` should typically be an optimizer that returns multiple final solutions, such as [moga](#) or [soga](#).
- The `model_pointer` keyword must identify a surrogate model which includes an underlying truth model, typically via `truth_model_pointer` or `dace_method_pointer`

Workflow:

- One might first try a single minimization method coupled with a surrogate model prior to using this surrogate-based global method. This is essentially equivalent to setting `max_iterations` to 1 and will allow one to get a sense of what surrogate types are the most accurate to use for the problem.
- Consider starting with a small number of maximum iterations, such as 3–5, to get a sense of how the optimization evolves as the surrogate gets updated. If it is still changing significantly, then a larger number (used in combination with restart) may be needed.
- Surrogates can be built for all primary functions and constraints or for only a subset of these functions and constraints. This allows one to use a "truth" model directly for some of the response functions, perhaps due to them being much less expensive than other functions.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

Theory

In surrogate-based optimization (SBO) and surrogate-based nonlinear least squares (SBNLS), minimization occurs using a set of one or more approximations, defined from a surrogate model, that are built and periodically updated using data from a "truth" model. The surrogate model can be a global data fit (e.g., regression or interpolation of data generated from a design of computer experiments), a multipoint approximation, a local Taylor Series expansion, or a model hierarchy approximation (e.g., a low-fidelity simulation model), whereas the truth model involves a high-fidelity simulation model. The goals of surrogate-based methods are to reduce the total number of truth model simulations and, in the case of global data fit surrogates, to smooth noisy data with an easily navigated analytic function.

The `surrogate_based_global` method was originally designed for MOGA (a multi-objective genetic algorithm). Since genetic algorithms often need thousands or tens of thousands of points to produce optimal or near-optimal solutions, the use of surrogates can be helpful for reducing the truth model evaluations. Instead of creating one set of surrogates for the individual objectives and running the optimization algorithm on the surrogate once, the idea is to select points along the (surrogate) Pareto frontier, which can be used to supplement the existing points.

In this way, one does not need to use many points initially to get a very accurate surrogate. The surrogate becomes more accurate as the iterations progress.

See Also

These keywords may also be of interest:

- [efficient_global](#)
- [surrogate_based_local](#)

7.2.9.1 `method_pointer`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_global](#)
- [method_pointer](#)

Pointer to sub-method to apply to a surrogate or branch-and-bound sub-problem

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: `approx_method_pointer`

Argument(s): STRING

Description

The `method_pointer` keyword is used to specify a pointer to an optimization or least-squares sub-method to apply in the context of what could be described as hierarchical methods. In surrogate-based methods, the sub-method is applied to the surrogate model. In the branch-and-bound method, the sub-method is applied to the relaxed sub-problems.

Any `model_pointer` identified in the sub-method specification is ignored. Instead, the parent method is responsible for selecting the appropriate model to use as specified by its `model_pointer`. In surrogate-based methods, it is a surrogate model defined using its `model_pointer`. In branch-and-bound methods, it is the relaxed model that is constructed internally from the original model.

7.2.9.2 `method_name`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_global](#)
- [method_name](#)

Specify sub-method by name

Specification

Alias: `approx_method_name`

Argument(s): STRING

Description

The `method_name` keyword is used to specify a sub-method by Dakota method name (e.g. 'npsol_sqp') rather than block pointer. The method will be executed using its default settings. The optional `model_pointer` specification can be used to associate a model block with the method.

7.2.9.3 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_global](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: `approx_model_pointer`

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.9.4 `replace_points`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_global](#)
- [replace_points](#)

(Recommended) Replace points in the surrogate training set, instead of appending

Specification

Alias: none

Argument(s): none

Default: Points appended, not replaced

Description

The user has the option of appending the optimal points from the surrogate model to the current set of truth points or using the optimal points from the surrogate model to replace the optimal set of points from the previous iteration. Although appending to the set is the default behavior, at this time we strongly recommend using the option `replace_points` because it appears to be more accurate and robust.

7.2.9.5 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_global](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.10 dot_frcg

- [Keywords Area](#)
- [method](#)
- [dot_frcg](#)

DOT conjugate gradient optimization method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence

	Optional	constraint_ - tolerance	Maximum allowable constraint violation still considered feasible
	Optional	speculative	Compute speculative gradients
	Optional	max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

DOT library [88] implementation of Fletcher-Reeves conjugate gradient method for unconstrained optimization.

See [package_dot](#) for information common to all DOT methods.

DOT requires a separate software license and therefore may not be available in all versions of Dakota. CONMIN or OPT++ methods may be suitable alternatives.

Caution regarding `dot_frcg`. In DOT Version 4.20, we have noticed inconsistent behavior of this algorithm across different versions of Linux. Our best assessment is that it is due to different treatments of uninitialized variables. As we do not know the intention of the code authors and maintaining DOT source code is outside of the Dakota project scope, we have not made nor are we recommending any code changes to address this. However, all users who use `dot_frcg` in DOT Version 4.20 should be aware that results may not be reliable.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Calibration](#) (when [calibration_terms](#) are specified)

7.2.10.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [dot_frcg](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.10.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [dot_frcg](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.10.3 `constraint_tolerance`

- [Keywords Area](#)
- [method](#)
- [dot_frcg](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.10.4 speculative

- [Keywords Area](#)
- [method](#)
- [dot_frcg](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical

or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (`dakota` numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by `Dakota` (it will be ignored for `vendor numerical` gradients).

7.2.10.5 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [dot_frcg](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.10.6 `scaling`

- [Keywords Area](#)
- [method](#)

- [dot_frcg](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];

- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.10.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [dot_frcg](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0.  0.
  upper_bounds = 1.  1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.11 dot_mmfd

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)

DOT modified method of feasible directions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		constraint_tolerance	Maximum allowable constraint violation still considered feasible

	Optional	speculative	Compute speculative gradients
	Optional	max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

DOT library [88] implementation of the modified method of feasible directions for constrained optimization.

See [package_dot](#) for information common to all DOT methods.

DOT requires a separate software license and therefore may not be available in all versions of Dakota. CONMIN or OPT++ methods may be suitable alternatives.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when [calibration_terms](#) are specified)

7.2.11.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.11.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.11.3 `constraint_tolerance`

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.11.4 speculative

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search

in user-supplied gradient mode (`dakota` numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor` numerical gradients).

7.2.11.5 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.11.6 `scaling`

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.11.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [dot_mmfd](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.12 dot_bfgs

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)

DOT BFGS optimization method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		constraint_tolerance	Maximum allowable constraint violation still considered feasible
	Optional		speculative	Compute speculative gradients
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers

	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

DOT library [88] implementation of Broyden-Fletcher-Goldfarb-Shanno method for unconstrained optimization.

See [package_dot](#) for information common to all DOT methods.

DOT requires a separate software license and therefore may not be available in all versions of Dakota. CONMIN or OPT++ methods may be suitable alternatives.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Calibration](#) (when [calibration_terms](#) are specified)

7.2.12.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.12.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.12.3 constraint_tolerance

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.12.4 speculative

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.12.5 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.12.6 scaling

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from `method`, these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.12.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [dot_bfgs](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.13 dot_slp

- [Keywords Area](#)
- [method](#)
- [dot_slp](#)

DOT Sequential Linear Program

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		constraint_tolerance	Maximum allowable constraint violation still considered feasible
	Optional		speculative	Compute speculative gradients
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

DOT library [88] implementation of sequential linear programming for constrained optimization.

See [package_dot](#) for information common to all DOT methods.

DOT requires a separate software license and therefore may not be available in all versions of Dakota. CONMIN or OPT++ methods may be suitable alternatives.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when [calibration_terms](#) are specified)

7.2.13.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [dot_slp](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.13.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [dot_slp](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.13.3 constraint_tolerance

- [Keywords Area](#)
- [method](#)

- [dot_slp](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.13.4 speculative

- [Keywords Area](#)
- [method](#)
- [dot_slp](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.13.5 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [dot_slp](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.13.6 scaling

- [Keywords Area](#)
- [method](#)
- [dot_slp](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the `variables` and `responses` blocks. When the `scaling` keyword is omitted from `method`, these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in `responses` for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.13.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [dot_slp](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.14 dot_sqp

- [Keywords Area](#)
- [method](#)
- [dot_sqp](#)

DOT Sequential Quadratic Program

Topics

This keyword is related to the topics:

- [package_dot](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		constraint_tolerance	Maximum allowable constraint violation still considered feasible
	Optional		speculative	Compute speculative gradients
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

DOT library [88] implementation of sequential quadratic programming for constrained optimization.

See [package_dot](#) for information common to all DOT methods.

DOT requires a separate software license and therefore may not be available in all versions of Dakota. CONMIN or OPT++ methods may be suitable alternatives.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)

- [Best Nonlinear Constraints](#)
- [Calibration](#) (when [calibration_terms](#) are specified)

7.2.14.1 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [dot_sqp](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.14.2 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [dot_sqp](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.14.3 `constraint_tolerance`

- [Keywords Area](#)
- [method](#)
- [dot_sqp](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.14.4 speculative

- [Keywords Area](#)
- [method](#)
- [dot_sqp](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a

trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.14.5 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [dot_sqp](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.14.6 scaling

- [Keywords Area](#)
- [method](#)
- [dot_sqp](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.14.7 `model_pointer`

- [Keywords Area](#)
- [method](#)

- [dot_sqp](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
```

```

model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.15 conmin_frcg

- [Keywords Area](#)
- [method](#)
- [conmin_frcg](#)

CONMIN conjugate gradient optimization method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence

	Optional	constraint_ - tolerance	Maximum allowable constraint violation still considered feasible
	Optional	speculative	Compute speculative gradients
	Optional	max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The interpretations of the method independent controls for CONMIN are essentially identical to those for DOT.

See [package_dot](#) for information related to CONMIN methods, specifically [dot_frcg](#).

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Calibration](#) (when [calibration_terms](#) are specified)

See Also

These keywords may also be of interest:

- [dot_frcg](#)

7.2.15.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [conmin_frcg](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.15.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [conmin_frcg](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.15.3 `constraint_tolerance`

- [Keywords Area](#)
- [method](#)
- [conmin_frcg](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.15.4 speculative

- [Keywords Area](#)
- [method](#)
- [conmin_frcg](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical

or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.15.5 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [conmin_frcg](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.15.6 `scaling`

- [Keywords Area](#)
- [method](#)

- [conmin_frcg](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];

- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.15.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [conmin_frcg](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```



```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.16 conmin_mfd

- [Keywords Area](#)
- [method](#)
- [conmin_mfd](#)

CONMIN method of feasible directions

Topics

This keyword is related to the topics:

- [package_conmin](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence

	Optional	constraint_ - tolerance	Maximum allowable constraint violation still considered feasible
	Optional	speculative	Compute speculative gradients
	Optional	max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The interpretations of the method independent controls for CONMIN are essentially identical to those for DOT.

See [package_dot](#) for information related to CONMIN methods, specifically [dot_mmfd](#).

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when [calibration_terms](#) are specified)

See Also

These keywords may also be of interest:

- [dot_mmfd](#)

7.2.16.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [conmin_mfd](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.16.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [conmin_mfd](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.16.3 `constraint_tolerance`

- [Keywords Area](#)
- [method](#)
- [conmin_mfd](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.16.4 speculative

- [Keywords Area](#)
- [method](#)
- [conmin_mfd](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical

or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.16.5 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [conmin_mfd](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.16.6 `scaling`

- [Keywords Area](#)
- [method](#)

- [conmin_mfd](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];

- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.16.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [conmin_mfd](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.17 dl_solver

- [Keywords Area](#)
- [method](#)
- [dl_solver](#)

(Experimental) Dynamically-loaded solver

Topics

This keyword is related to the topics:

- [optimization_and_calibration](#)

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

This keyword specifies a dynamically-loaded optimization solver library, an experimental Dakota feature that is not enabled by default.

7.2.17.1 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [dl_solver](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.17.2 `scaling`

- [Keywords Area](#)
- [method](#)
- [dl_solver](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from `method`, these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.17.3 model_pointer

- [Keywords Area](#)
- [method](#)
- [dl_solver](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.18 npsol_sqp

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)

NPSOL Sequential Quadratic Program

Topics

This keyword is related to the topics:

- [package_npsol](#)
- [sequential_quadratic_programming](#)
- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		verify_level	Verify the quality of analytic gradients
	Optional		function_precision	Specify the maximum precision of the analysis code responses
	Optional		linesearch_tolerance	Choose how accurately the algorithm will compute the minimum in a line search
	Optional		convergence_tolerance	Stopping criterion based on objective function convergence
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods

	Optional	<code>constraint_tolerance</code>	Maximum allowable constraint violation still considered feasible
	Optional	<code>speculative</code>	Compute speculative gradients
	Optional	<code>max_function_evaluations</code>	Number of function evaluations allowed for optimizers
	Optional	<code>scaling</code>	Turn on scaling for variables, responses, and constraints
	Optional	<code>model_pointer</code>	Identifier for model block to be used by a method

Description

Sequential quadratic programming optimizer.

NPSOL requires a separate software license and therefore may not be available in all versions of Dakota. CONMIN or OPT++ methods may be suitable alternatives.

Stopping Criteria

The method independent controls for `max_iterations` and `max_function_evaluations` limit the number of major SQP iterations and the number of function evaluations that can be performed during an NPSOL optimization. The `convergence_tolerance` control defines NPSOL's internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function). The `constraint_tolerance` control defines how tightly the constraint functions are satisfied at convergence. The default value is dependent upon the machine precision of the platform in use, but is typically on the order of `1.e-8` for double precision computations. Extremely small values for `constraint_tolerance` may not be attainable. The `output_verbosity` setting controls the amount of information generated at each major SQP iteration: the `silent` and `quiet` settings result in only one line of diagnostic output for each major iteration and print the final optimization solution, whereas the `verbose` and `debug` settings add additional information on the objective function, constraints, and variables at each major iteration.

Concurrency

NPSOL is not a parallel algorithm and cannot directly take advantage of concurrent evaluations. However, if `numerical_gradients` with `method_source dakota` is specified, then the finite difference function evaluations can be performed concurrently (using any of the parallel modes described in the Users Manual [5]).

An important related observation is the fact that NPSOL uses two different line searches depending on how gradients are computed. For either `analytic_gradients` or `numerical_gradients` with `method_source dakota`, NPSOL is placed in user-supplied gradient mode (NPSOL's "Derivative Level" is set to 3) and it uses a gradient-based line search (the assumption is that user-supplied gradients are inexpensive). On the other hand, if `numerical_gradients` are selected with `method_source vendor`, then NPSOL is computing finite differences internally and it will use a value-based line search (the assumption is that finite differencing on each line search evaluation is too expensive). The ramifications of this are: (1) performance will vary between `method_source dakota` and `method_source vendor` for `numerical_gradients`, and (2) gradient speculation is unnecessary when performing optimization in parallel since the gradient-based line search in user-supplied gradient mode is already load balanced for parallel execution. Therefore, a `speculative` specification will be ignored

by NPSOL, and optimization with numerical gradients should select `method_source dakota` for load balanced parallel operation and `method_source vendor` for efficient serial operation.

Linear constraints

Lastly, NPSOL supports specialized handling of linear inequality and equality constraints. By specifying the coefficients and bounds of the linear inequality constraints and the coefficients and targets of the linear equality constraints, this information can be provided to NPSOL at initialization and tracked internally, removing the need for the user to provide the values of the linear constraints on every function evaluation.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

7.2.18.1 `verify_level`

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [verify_level](#)

Verify the quality of analytic gradients

Specification

Alias: none

Argument(s): INTEGER

Default: -1 (no gradient verification)

Description

`verify_level` instructs the NPSOL and NLSSOL algorithms to perform their own finite difference verification of the gradients provided by Dakota. Typically these are used to verify `analytic_gradients` produced by a simulation code, though the option can be used with other Dakota-supplied gradient types including numerical or mixed.

Level 1 will verify the objective gradients, level 2, the nonlinear constraint gradients, and level 3, both. See the Optional Input Parameters section of the NPSOL manual [33] for additional information, including options to verify at the user-supplied initial point vs. first feasible point.

7.2.18.2 `function_precision`

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)

- [function_precision](#)

Specify the maximum precision of the analysis code responses

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

The `function_precision` control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.

7.2.18.3 `linesearch_tolerance`

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [linesearch_tolerance](#)

Choose how accurately the algorithm will compute the minimum in a line search

Specification

Alias: none

Argument(s): REAL

Default: 0.9 (inaccurate line search)

Description

The `linesearch_tolerance` setting controls the accuracy of the line search. The smaller the value (between 0 and 1), the more accurately the algorithm will attempt to compute a precise minimum along the search direction.

7.2.18.4 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function convergence

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

NPSOL defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.0e-6` will result in approximately six digits of accuracy in the final objective function).

7.2.18.5 max_iterations

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.18.6 `constraint_tolerance`

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Dakota's `constraint_tolerance` settings maps to NPSOL's feasibility tolerance which is an absolute tolerance on the constraints.

7.2.18.7 `speculative`

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.18.8 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.18.9 scaling

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the `variables` and `responses` blocks. When the `scaling` keyword is omitted from `method`, these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in `responses` for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.18.10 model_pointer

- [Keywords Area](#)
- [method](#)
- [npsol_sqp](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.19 nlssol_sqp

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)

Sequential Quadratic Program for nonlinear least squares

Topics

This keyword is related to the topics:

- [sequential_quadratic_programming](#)
- [nonlinear_least_squares](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		verify_level	Verify the quality of analytic gradients
	Optional		function_precision	Specify the maximum precision of the analysis code responses
	Optional		linsearch_tolerance	Choose how accurately the algorithm will compute the minimum in a line search
	Optional		convergence_tolerance	Stopping criterion based on objective function convergence
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		constraint_tolerance	Maximum allowable constraint violation still considered feasible
	Optional		speculative	Compute speculative gradients
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

NLSSOL supports unconstrained, bound-constrained, and generally-constrained least-squares calibration problems. It exploits the structure of a least squares objective function through the periodic use of Gauss-Newton Hessian approximations to accelerate the SQP algorithm.

NLSSOL requires a separate software license and therefore may not be available in all versions of Dakota. [nl2sol](#) or [optpp_g_newton](#) may be suitable alternatives.

Stopping Criteria

The method independent controls for `max_iterations` and `max_function_evaluations` limit the number of major SQP iterations and the number of function evaluations that can be performed during an NPSOL optimization. The `convergence_tolerance` control defines NPSOL's internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.e-6` will result in approximately six digits of accuracy in the final objective function). The `constraint_tolerance` control defines how tightly the constraint functions are satisfied at convergence. The default value is dependent upon the machine precision of the platform in use, but is typically on the order of `1.e-8` for double precision computations. Extremely small values for `constraint_tolerance` may not be attainable.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)
- [Parameter Confidence Intervals](#) (when `num_experiments` equals 1)

See Also

These keywords may also be of interest:

- [npsol_sqp](#)
- [nl2sol](#)
- [optpp_g_newton](#)
- [field_calibration_terms](#)

7.2.19.1 `verify_level`

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [verify_level](#)

Verify the quality of analytic gradients

Specification

Alias: none

Argument(s): INTEGER

Default: -1 (no gradient verification)

Description

`verify_level` instructs the NPSOL and NLSSOL algorithms to perform their own finite difference verification of the gradients provided by Dakota. Typically these are used to verify `analytic_gradients` produced by a simulation code, though the option can be used with other Dakota-supplied gradient types including numerical or mixed.

Level 1 will verify the objective gradients, level 2, the nonlinear constraint gradients, and level 3, both. See the Optional Input Parameters section of the NPSOL manual [33] for additional information, including options to verify at the user-supplied initial point vs. first feasible point.

7.2.19.2 `function_precision`

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [function_precision](#)

Specify the maximum precision of the analysis code responses

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

The `function_precision` control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.

7.2.19.3 `linesearch_tolerance`

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [linesearch_tolerance](#)

Choose how accurately the algorithm will compute the minimum in a line search

Specification

Alias: none

Argument(s): REAL

Default: 0.9 (inaccurate line search)

Description

The `linesearch_tolerance` setting controls the accuracy of the line search. The smaller the value (between 0 and 1), the more accurately the algorithm will attempt to compute a precise minimum along the search direction.

7.2.19.4 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function convergence

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

NPSOL defines an internal optimality tolerance which is used in evaluating if an iterate satisfies the first-order Kuhn-Tucker conditions for a minimum. The magnitude of `convergence_tolerance` approximately specifies the number of significant digits of accuracy desired in the final objective function (e.g., `convergence_tolerance = 1.0e-6` will result in approximately six digits of accuracy in the final objective function).

7.2.19.5 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.19.6 constraint_tolerance

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Dakota's `constraint_tolerance` settings maps to NPSOL's feasibility tolerance which is an absolute tolerance on the constraints.

7.2.19.7 speculative

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.19.8 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.19.9 `scaling`

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.19.10 model_pointer

- [Keywords Area](#)
- [method](#)
- [nlssol_sqp](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.20 nlpql_sqp

- [Keywords Area](#)
- [method](#)
- [nlpql_sqp](#)

NLPQL Sequential Quadratic Program

Topics

This keyword is related to the topics:

- [package_nlpql](#)
- [sequential_quadratic_programming](#)
- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers

	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

NLPQL implementation of sequential quadratic programming. The particular SQP implementation in `nlpql_sqp` uses a variant with distributed and non-monotone line search. Thus, this variant is designed to be more robust in the presence of inaccurate or noisy gradients common in many engineering applications.

NLPQL requires a separate software license and therefore may not be available in all versions of Dakota. CONMIN or OPT++ methods may be suitable alternatives.

The method independent controls for maximum iterations and output verbosity are mapped to NLPQL controls MAX-IT and IPRINT, respectively. The maximum number of function evaluations is enforced within the NLPQLOptimizer class.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

7.2.20.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [nlpql_sqp](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.20.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [nlpql_sqp](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.20.3 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [nlpql_sqp](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.20.4 scaling

- [Keywords Area](#)
- [method](#)
- [nlpql_sqp](#)

- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0, 1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;

- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100

responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.20.5 model_pointer

- [Keywords Area](#)
- [method](#)
- [nlpql_sqp](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.21 optpp_cg

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)

A conjugate gradient optimization method

Topics

This keyword is related to the topics:

- [package_optpp](#)
- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_step	Max change in design point
	Optional		gradient_tolerance	Stopping criteria based on L2 norm of gradient
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence

	Optional	speculative	Compute speculative gradients
	Optional	max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The conjugate gradient method is an implementation of the Polak-Ribiere approach and handles only unconstrained problems.

See [package_optpp](#) for info related to all `optpp` methods.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Calibration](#) (when [calibration_terms](#) are specified)

See Also

These keywords may also be of interest:

- [optpp_g_newton](#)
- [optpp_pds](#)
- [optpp_fd_newton](#)
- [optpp_newton](#)
- [optpp_g_newton](#)

7.2.21.1 max_step

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)
- [max_step](#)

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The `max_step` control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The `gradient_tolerance` control is defined for all gradient-based optimizers.

7.2.21.2 `gradient_tolerance`

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)
- [gradient_tolerance](#)

Stopping criteria based on L2 norm of gradient

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient (for unconstrained minimum or no active constraints) or the Lagrangian gradient (for constrained minimum) that indicates convergence to a stationary point. The `gradient_tolerance` control is currently defined only for the OPT++ and ROL libraries.

7.2.21.3 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.21.4 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.21.5 `speculative`

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based

optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.21.6 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.21.7 scaling

- [Keywords Area](#)
- [method](#)
- [optpp_cg](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.21.8 `model_pointer`

- [Keywords Area](#)
- [method](#)

- [optpp_cg](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
```

```

    model_pointer = 'DACE_M'
    sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.22 optpp_q_newton

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)

Quasi-Newton optimization method

Topics

This keyword is related to the topics:

- [package_optpp](#)
- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		search_method	Select a search method for Newton-based optimizers

	Optional	merit_function	Balance goals of reducing objective function and satisfying constraints
	Optional	steplength_to_boundary	Controls how close to the boundary of the feasible region the algorithm is allowed to move
	Optional	centering_parameter	Controls how closely the algorithm should follow the "central path"
	Optional	max_step	Max change in design point
	Optional	gradient_tolerance	Stopping criteria based on L2 norm of gradient
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	speculative	Compute speculative gradients
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

This is a Newton method that expects a gradient and computes a low-rank approximation to the Hessian. Each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a nonlinear interior-point approach to manage the constraints.

See [package_optpp](#) for info related to all `optpp` methods.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

7.2.22.1 `search_method`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [search_method](#)

Select a search method for Newton-based optimizers

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Line Search Method (Group 1)	value_based_line_search	Use only function values for line search
			gradient_based_line_search	Set the search method to use the gradient
			trust_region	Use trust region as the globalization strategy.
			tr_pds	Use direct search as the local search in a trust region method

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.22.1.1 `value_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [search_method](#)
- [value_based_line_search](#)

Use only function values for line search

Specification

Alias: none

Argument(s): none

Default: `trust_region` (unconstrained), `value_based_line_search` (bound/general constraints)

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.22.1.2 `gradient_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [search_method](#)
- [gradient_based_line_search](#)

Set the search method to use the gradient

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.22.1.3 `trust_region`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [search_method](#)
- [trust_region](#)

Use trust region as the globalization strategy.

Specification

Alias: none

Argument(s): none

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.22.1.4 `tr_pds`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [search_method](#)
- [tr_pds](#)

Use direct search as the local search in a trust region method

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.22.2 merit_function

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [merit_function](#)

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none

Argument(s): none

Default: `argaez_tapia`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Merit Function (Group 1)	el_bakry	El-Bakry merit function
			argaez_tapia	The merit function by Tapia and Argaez
			van_shanno	The merit function by Vanderbei and Shanno

Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

7.2.22.2.1 el_bakry

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [merit_function](#)
- [el_bakry](#)

El-Bakry merit function

Specification

Alias: none

Argument(s): none

Description

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is $n+1$ function evaluations. For more information, see [20].

7.2.22.2.2 argaez_tapia

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [merit_function](#)
- [argaez_tapia](#)

The merit function by Tapia and Argaez

Specification

Alias: none

Argument(s): none

Description

The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [83].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.22.2.3 `van_shanno`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [merit_function](#)
- [van_shanno](#)

The merit function by Vanderbei and Shanno

Specification

Alias: none

Argument(s): none

Description

The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see [85].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.22.3 `stplength_to_boundary`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [stplength_to_boundary](#)

Controls how close to the boundary of the feasible region the algorithm is allowed to move

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.8 (`el_bakry`), 0.99995 (`argaez_tapia`), 0.95 (`van_shanno`)

Description

The `stplength_to_boundary` specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.22.4 centering_parameter

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [centering_parameter](#)

Controls how closely the algorithm should follow the "central path"

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.2 (el_bakry), 0.2 (argaez_tapia), 0.1 (van_shanno)

Description

The `centering_parameter` specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the "central path". See [92] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.22.5 max_step

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [max_step](#)

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The `max_step` control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The `gradient_tolerance` control is defined for all gradient-based optimizers.

7.2.22.6 `gradient_tolerance`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [gradient_tolerance](#)

Stopping critiera based on L2 norm of gradient

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient (for unconstrained minimum or no active constraints) or the Lagrangian gradient (for constrained minimum) that indicates convergence to a stationary point. The `gradient_tolerance` control is currently defined only for the OPT++ and ROL libraries.

7.2.22.7 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.22.8 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.22.9 speculative

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information

on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical gradients`).

7.2.22.10 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.22.11 `scaling`

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)

- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0, 1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;

- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100

responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.22.12 model_pointer

- [Keywords Area](#)
- [method](#)
- [optpp_q_newton](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.23 optpp_fd_newton

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)

Finite Difference Newton optimization method

Topics

This keyword is related to the topics:

- [package_optpp](#)
- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		search_method	Select a search method for Newton-based optimizers
	Optional		merit_function	Balance goals of reducing objective function and satisfying constraints
	Optional		steplength_to_-boundary	Controls how close to the boundary of the feasible region the algorithm is allowed to move
	Optional		centering_-parameter	Controls how closely the algorithm should follow the "central path"

	Optional	max_step	Max change in design point
	Optional	gradient_tolerance	Stopping criteria based on L2 norm of gradient
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	speculative	Compute speculative gradients
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

This is a Newton method that expects a gradient and computes a finite-difference approximation to the Hessian. Each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a nonlinear interior-point approach to manage the constraints.

See [package_optpp](#) for info related to all `optpp` methods.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

See Also

These keywords may also be of interest:

- [optpp_cg](#)
- [optpp_g_newton](#)
- [optpp_pds](#)
- [optpp_newton](#)
- [optpp_g_newton](#)

7.2.23.1 search_method

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [search_method](#)

Select a search method for Newton-based optimizers

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Line Search Method (Group 1)	value_based_line_search	Use only function values for line search
			gradient_based_line_search	Set the search method to use the gradient
			trust_region	Use trust region as the globalization strategy.
			tr_pds	Use direct search as the local search in a trust region method

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.23.1.1 `value_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [search_method](#)
- [value_based_line_search](#)

Use only function values for line search

Specification

Alias: none

Argument(s): none

Default: `trust_region` (unconstrained), `value_based_line_search` (bound/general constraints)

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.23.1.2 `gradient_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [search_method](#)
- [gradient_based_line_search](#)

Set the search method to use the gradient

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.23.1.3 `trust_region`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [search_method](#)
- [trust_region](#)

Use trust region as the globalization strategy.

Specification

Alias: none

Argument(s): none

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.23.1.4 `tr_pds`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [search_method](#)
- [tr_pds](#)

Use direct search as the local search in a trust region method

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.23.2 merit_function

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [merit_function](#)

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none

Argument(s): none

Default: `argaez_tapia`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Merit Function (Group 1)	el_bakry	El-Bakry merit function
			argaez_tapia	The merit function by Tapia and Argaez
			van_shanno	The merit function by Vanderbei and Shanno

Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

7.2.23.2.1 `el_bakry`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [merit_function](#)
- [el_bakry](#)

El-Bakry merit function

Specification

Alias: none

Argument(s): none

Description

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is $n+1$ function evaluations. For more information, see [20].

7.2.23.2.2 `argaez_tapia`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [merit_function](#)
- [argaez_tapia](#)

The merit function by Tapia and Argaez

Specification

Alias: none

Argument(s): none

Description

The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [83].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.23.2.3 `van_shanno`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [merit_function](#)
- [van_shanno](#)

The merit function by Vanderbei and Shanno

Specification

Alias: none

Argument(s): none

Description

The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see [85].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.23.3 `stplength_to_boundary`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [stplength_to_boundary](#)

Controls how close to the boundary of the feasible region the algorithm is allowed to move

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.8 (`el_bakry`), 0.99995 (`argaez_tapia`), 0.95 (`van_shanno`)

Description

The `stplength_to_boundary` specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.23.4 centering_parameter

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [centering_parameter](#)

Controls how closely the algorithm should follow the "central path"

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.2 (el_bakry), 0.2 (argaez_tapia), 0.1 (van_shanno)

Description

The `centering_parameter` specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the "central path". See [92] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.23.5 max_step

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [max_step](#)

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The `max_step` control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The `gradient_tolerance` control is defined for all gradient-based optimizers.

7.2.23.6 `gradient_tolerance`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [gradient_tolerance](#)

Stopping critiera based on L2 norm of gradient

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient (for unconstrained minimum or no active constraints) or the Lagrangian gradient (for constrained minimum) that indicates convergence to a stationary point. The `gradient_tolerance` control is currently defined only for the OPT++ and ROL libraries.

7.2.23.7 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.23.8 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.23.9 speculative

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information

on these specifications). It should not be selected with `vendor numerical gradients` since `vendor` internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (`dakota numerical` or `analytic gradients`) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical gradients`).

7.2.23.10 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.23.11 `scaling`

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)

- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0, 1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;

- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.23.12 model_pointer

- [Keywords Area](#)
- [method](#)
- [optpp_fd_newton](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.24 optpp_g_newton

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)

Newton method based least-squares calibration

Topics

This keyword is related to the topics:

- [package_optpp](#)
- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		search_method	Select a search method for Newton-based optimizers
	Optional		merit_function	Balance goals of reducing objective function and satisfying constraints
	Optional		steplength_to_-boundary	Controls how close to the boundary of the feasible region the algorithm is allowed to move
	Optional		centering_-parameter	Controls how closely the algorithm should follow the "central path"

	Optional	max_step	Max change in design point
	Optional	gradient_tolerance	Stopping criteria based on L2 norm of gradient
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	speculative	Compute speculative gradients
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The Gauss-Newton algorithm is available as `optpp_g_newton` and supports unconstrained, bound-constrained, and generally-constrained problems. When interfaced with the unconstrained, bound-constrained, and nonlinear interior point full-Newton optimizers from the OPT++ library, it provides a Gauss-Newton least squares capability which – on zero-residual test problems – can exhibit quadratic convergence rates near the solution. (Real problems almost never have zero residuals, i.e., perfect fits.)

See [package_optpp](#) for info related to all `optpp` methods.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)
- [Parameter Confidence Intervals](#) (when `num_experiments` equals 1)

See Also

These keywords may also be of interest:

- [optpp_cg](#)
- [optpp_pds](#)
- [optpp_fd_newton](#)
- [optpp_newton](#)
- [optpp_g_newton](#)
- [field_calibration_terms](#)

7.2.24.1 search_method

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [search_method](#)

Select a search method for Newton-based optimizers

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Line Search Method (Group 1)	value_based_line_search	Use only function values for line search
			gradient_based_line_search	Set the search method to use the gradient
			trust_region	Use trust region as the globalization strategy.
			tr_pds	Use direct search as the local search in a trust region method

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.24.1.1 `value_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [search_method](#)
- [value_based_line_search](#)

Use only function values for line search

Specification

Alias: none

Argument(s): none

Default: `trust_region` (unconstrained), `value_based_line_search` (bound/general constraints)

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.24.1.2 `gradient_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [search_method](#)
- [gradient_based_line_search](#)

Set the search method to use the gradient

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.24.1.3 `trust_region`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [search_method](#)
- [trust_region](#)

Use trust region as the globalization strategy.

Specification

Alias: none

Argument(s): none

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.24.1.4 `tr_pds`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [search_method](#)
- [tr_pds](#)

Use direct search as the local search in a trust region method

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.24.2 merit_function

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [merit_function](#)

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none

Argument(s): none

Default: `argaez_tapia`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Merit Function (Group 1)	el_bakry	El-Bakry merit function
			argaez_tapia	The merit function by Tapia and Argaez
			van_shanno	The merit function by Vanderbei and Shanno

Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

7.2.24.2.1 `el_bakry`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [merit_function](#)
- [el_bakry](#)

El-Bakry merit function

Specification

Alias: none

Argument(s): none

Description

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is $n+1$ function evaluations. For more information, see [20].

7.2.24.2.2 `argaez_tapia`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [merit_function](#)
- [argaez_tapia](#)

The merit function by Tapia and Argaez

Specification

Alias: none

Argument(s): none

Description

The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [83].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.24.2.3 `van_shanno`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [merit_function](#)
- [van_shanno](#)

The merit function by Vanderbei and Shanno

Specification

Alias: none

Argument(s): none

Description

The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see [85].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.24.3 `stplength_to_boundary`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [stplength_to_boundary](#)

Controls how close to the boundary of the feasible region the algorithm is allowed to move

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.8 (`el_bakry`), 0.99995 (`argaez_tapia`), 0.95 (`van_shanno`)

Description

The `stplength_to_boundary` specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.24.4 centering_parameter

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [centering_parameter](#)

Controls how closely the algorithm should follow the "central path"

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.2 (el_bakry), 0.2 (argaez_tapia), 0.1 (van_shanno)

Description

The `centering_parameter` specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the "central path". See [92] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.24.5 max_step

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [max_step](#)

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The `max_step` control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The `gradient_tolerance` control is defined for all gradient-based optimizers.

7.2.24.6 `gradient_tolerance`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [gradient_tolerance](#)

Stopping critiera based on L2 norm of gradient

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient (for unconstrained minimum or no active constraints) or the Lagrangian gradient (for constrained minimum) that indicates convergence to a stationary point. The `gradient_tolerance` control is currently defined only for the OPT++ and ROL libraries.

7.2.24.7 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.24.8 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.24.9 speculative

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information

on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical gradients`).

7.2.24.10 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.24.11 `scaling`

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)

- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0, 1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;

- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100

responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.24.12 model_pointer

- [Keywords Area](#)
- [method](#)
- [optpp_g_newton](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.25 optpp_newton

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)

Newton method based optimization

Topics

This keyword is related to the topics:

- [package_optpp](#)
- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		search_method	Select a search method for Newton-based optimizers
	Optional		merit_function	Balance goals of reducing objective function and satisfying constraints
	Optional		steplength_to_-boundary	Controls how close to the boundary of the feasible region the algorithm is allowed to move
	Optional		centering_-parameter	Controls how closely the algorithm should follow the "central path"

	Optional	max_step	Max change in design point
	Optional	gradient_tolerance	Stopping criteria based on L2 norm of gradient
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	speculative	Compute speculative gradients
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

This is a full Newton method that expects a gradient and a Hessian. Each of the Newton-based methods are automatically bound to the appropriate OPT++ algorithm based on the user constraint specification (unconstrained, bound-constrained, or generally-constrained). In the generally-constrained case, the Newton methods use a non-linear interior-point approach to manage the constraints.

See [package_optpp](#) for info related to all `optpp` methods.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

See Also

These keywords may also be of interest:

- [optpp_cg](#)
- [optpp_g_newton](#)
- [optpp_pds](#)
- [optpp_fd_newton](#)
- [optpp_g_newton](#)

7.2.25.1 `search_method`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [search_method](#)

Select a search method for Newton-based optimizers

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Line Search Method (Group 1)	value_based_line_search	Use only function values for line search
			gradient_based_line_search	Set the search method to use the gradient
			trust_region	Use trust region as the globalization strategy.
			tr_pds	Use direct search as the local search in a trust region method

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.25.1.1 `value_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [search_method](#)
- [value_based_line_search](#)

Use only function values for line search

Specification

Alias: none

Argument(s): none

Default: `trust_region` (unconstrained), `value_based_line_search` (bound/general constraints)

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.25.1.2 `gradient_based_line_search`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [search_method](#)
- [gradient_based_line_search](#)

Set the search method to use the gradient

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.25.1.3 `trust_region`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [search_method](#)
- [trust_region](#)

Use trust region as the globalization strategy.

Specification

Alias: none

Argument(s): none

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.25.1.4 `tr_pds`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [search_method](#)
- [tr_pds](#)

Use direct search as the local search in a trust region method

Specification

Alias: none

Argument(s): none

Description

The `search_method` control is defined for all Newton-based optimizers and is used to select between `trust_region`, `gradient_based_line_search`, and `value_based_line_search` methods. The `gradient_based_line_search` option uses the line search method proposed by [63]. This option satisfies sufficient decrease and curvature conditions; whereas, `value_base_line_search` only satisfies the sufficient decrease condition. At each line search iteration, the `gradient_based_line_search` method computes the function and gradient at the trial point. Consequently, given expensive function evaluations, the `value_based_line_search` method is preferred to the `gradient_based_line_search` method. Each of these Newton methods additionally supports the `tr_pds` selection for unconstrained problems. This option performs a robust trust region search using pattern search techniques. Use of a line search is the default for bound-constrained and generally-constrained problems, and use of a `trust_region` search method is the default for unconstrained problems.

7.2.25.2 merit_function

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [merit_function](#)

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none

Argument(s): none

Default: `argaez_tapia`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Merit Function (Group 1)	el_bakry	El-Bakry merit function
			argaez_tapia	The merit function by Tapia and Argaez
			van_shanno	The merit function by Vanderbei and Shanno

Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

7.2.25.2.1 `el_bakry`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [merit_function](#)
- [el_bakry](#)

El-Bakry merit function

Specification

Alias: none

Argument(s): none

Description

The "el_bakry" merit function is the L2-norm of the first order optimality conditions for the nonlinear programming problem. The cost per linesearch iteration is $n+1$ function evaluations. For more information, see [20].

7.2.25.2.2 `argaez_tapia`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [merit_function](#)
- [argaez_tapia](#)

The merit function by Tapia and Argaez

Specification

Alias: none

Argument(s): none

Description

The "argaez_tapia" merit function can be classified as a modified augmented Lagrangian function. The augmented Lagrangian is modified by adding to its penalty term a potential reduction function to handle the perturbed complementarity condition. The cost per linesearch iteration is one function evaluation. For more information, see [83].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.25.2.3 `van_shanno`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [merit_function](#)
- [van_shanno](#)

The merit function by Vanderbei and Shanno

Specification

Alias: none

Argument(s): none

Description

The "van_shanno" merit function can be classified as a penalty function for the logarithmic barrier formulation of the nonlinear programming problem. The cost per linesearch iteration is one function evaluation. For more information see [85].

If the function evaluation is expensive or noisy, set the `merit_function` to "argaez_tapia" or "van_shanno".

7.2.25.3 `stplength_to_boundary`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [stplength_to_boundary](#)

Controls how close to the boundary of the feasible region the algorithm is allowed to move

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.8 (`el_bakry`), 0.99995 (`argaez_tapia`), 0.95 (`van_shanno`)

Description

The `stplength_to_boundary` specification is a parameter (between 0 and 1) that controls how close to the boundary of the feasible region the algorithm is allowed to move. A value of 1 means that the algorithm is allowed to take steps that may reach the boundary of the feasible region. If the user wishes to maintain strict feasibility of the design parameters this value should be less than 1. Default values are .8, .99995, and .95 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.25.4 `centering_parameter`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [centering_parameter](#)

Controls how closely the algorithm should follow the "central path"

Specification

Alias: none

Argument(s): REAL

Default: Merit function dependent: 0.2 (`el_bakry`), 0.2 (`argaez_tapia`), 0.1 (`van_shanno`)

Description

The `centering_parameter` specification is a parameter (between 0 and 1) that controls how closely the algorithm should follow the "central path". See [92] for the definition of central path. The larger the value, the more closely the algorithm follows the central path, which results in small steps. A value of 0 indicates that the algorithm will take a pure Newton step. Default values are .2, .2, and .1 for the `el_bakry`, `argaez_tapia`, and `van_shanno` merit functions, respectively.

7.2.25.5 `max_step`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [max_step](#)

Max change in design point

Specification

Alias: none

Argument(s): REAL

Default: 1000.

Description

The `max_step` control specifies the maximum step that can be taken when computing a change in the current design point (e.g., limiting the Newton step computed from current gradient and Hessian information). It is equivalent to a move limit or a maximum trust region size. The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient that indicates convergence to an unconstrained minimum (no active constraints). The `gradient_tolerance` control is defined for all gradient-based optimizers.

7.2.25.6 `gradient_tolerance`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [gradient_tolerance](#)

Stopping critiera based on L2 norm of gradient

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient (for unconstrained minimum or no active constraints) or the Lagrangian gradient (for constrained minimum) that indicates convergence to a stationary point. The `gradient_tolerance` control is currently defined only for the OPT++ and ROL libraries.

7.2.25.7 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.25.8 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.25.9 speculative

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information

on these specifications). It should not be selected with `vendor numerical gradients` since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (`dakota numerical` or `analytic gradients`) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical gradients`).

7.2.25.10 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.25.11 `scaling`

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)

- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0, 1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;

- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.25.12 model_pointer

- [Keywords Area](#)
- [method](#)
- [optpp_newton](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```
analysis_driver = 'text_book'  
  
responses  
response_functions = 3  
no_gradients  
no_hessians
```

7.2.26 optpp_pds

- [Keywords Area](#)
- [method](#)
- [optpp_pds](#)

Simplex-based derivative free optimization method

Topics

This keyword is related to the topics:

- [package_optpp](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		search_scheme_size	Number of points to be used in the direct search template
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

The direct search algorithm, PDS (parallel direct search method), supports bound constraints.

The PDS method can directly exploit asynchronous evaluations; however, this capability has not yet been implemented in Dakota.

See [package_optpp](#) for info related to all `optpp` methods.

See Also

These keywords may also be of interest:

- [optpp_cg](#)
- [optpp_g_newton](#)
- [optpp_fd_newton](#)
- [optpp_newton](#)
- [optpp_g_newton](#)

7.2.26.1 search_scheme_size

- [Keywords Area](#)

- [method](#)
- [optpp_pds](#)
- [search_scheme_size](#)

Number of points to be used in the direct search template

Specification

Alias: none

Argument(s): INTEGER

Default: 32

Description

The `search_scheme_size` is defined for the PDS method to specify the number of points to be used in the direct search template.

7.2.26.2 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [optpp_pds](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.26.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [optpp_pds](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.26.4 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [optpp_pds](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.26.5 scaling

- [Keywords Area](#)
- [method](#)
- [optpp_pds](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.26.6 model_pointer

- [Keywords Area](#)
- [method](#)
- [optpp_pds](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.27 demo_tpl

- [Keywords Area](#)
- [method](#)
- [demo_tpl](#)

Third-party optimization library integration demonstration.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		variable_tolerance	Step length-based stopping criteria for derivative-free optimizers
	Optional		solution_target	Stopping criteria based on objective function value
	Optional		options_file	Filename for a YAML file that specifies Gaussian process options

Description

This is not a method that is intended for use. Rather it is part of a documented, functional example of how to integrate a third-party optimization library. It takes an initial point, generates a list of random parameter values, evaluates the function at each point in the list, and returns the point with the lowest function value.

Documented instructions on how to integrate a third-party library, working source code, and build infrastructure can be found in `$dakotaSource/packages/external/demo_tpl`. Additionally, integration instructions can be found in the Developers Manual.

7.2.27.1 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [demo_tpl](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.27.2 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [demo_tpl](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.27.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [demo_tpl](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.27.4 `variable_tolerance`

- [Keywords Area](#)
- [method](#)
- [demo_tpl](#)
- [variable_tolerance](#)

Step length-based stopping criteria for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 0.01

Description

The `variable_tolerance` keyword defines the minimum step length allowed by the optimizer and is used to determine convergence. It is applicable to `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`.

Default Behavior

The default value varies according to method as follows:

- `asynch_pattern_search`: 1.0e-2
- `coliny_cobyala`: 1.0e-4
- `coliny_pattern_search`: 1.0e-5
- `coliny_solis_wets`: 1.0e-6

- `mesh_adaptive_search`: 1.0e-6

Usage Tips

It is recommended that `variable_tolerance` be set to a value for which changes of that scale in parameter values cause negligible changes in the objective function.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    contraction_factor = 0.25
    variable_tolerance = 1.e-4
    solution_target = 1.e-6
    max_function_evaluations 500
    constraint_tolerance 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    variable_tolerance = 0.01
    seed = 1234
```

7.2.27.5 `solution_target`

- [Keywords Area](#)
- [method](#)
- [demo_tpl](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: no target

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.27.6 options_file

- [Keywords Area](#)
- [method](#)
- [demo_tpl](#)
- [options_file](#)

Filename for a YAML file that specifies Gaussian process options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the Gaussian process that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: gp_options.yaml

GP Parameters:
  Sigma Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  scaler name: standardization
  num restarts: 20
  Nugget:
    fixed nugget: 0.0
    estimate nugget: true
  Bounds:
    lower bound: 3.17e-8
    upper bound: 1.0e-2
  gp seed: 42
  standardize response: false
  Trend:
    estimate trend: true
  Options:
    max degree: 2
    reduced basis: false
    p-norm: 1.0
    scaler type: none
    regression solver type: SVD
    verbosity: 1
  kernel type: squared exponential
  Length-scale Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  verbosity: 1
```

7.2.28 rol

- [Keywords Area](#)

- [method](#)
- [rol](#)

Rapid Optimization Library (ROL) is a large-scale optimization package within Trilinos.

Topics

This keyword is related to the topics:

- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		variable_tolerance	Step length-based stopping criteria for derivative-free optimizers
	Optional		gradient_tolerance	Stopping criteria based on L2 norm of gradient
	Optional		constraint_tolerance	Maximum allowable constraint violation still considered feasible
	Optional		options_file	File containing advanced ROL options
	Optional		scaling	Turn on scaling for variables, responses, and constraints

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

ROL is used for the solution of optimal design, optimal control and inverse problems in large-scale engineering applications.

Usage Tips

ROL is a general gradient-based library designed to scale well to very large problem sizes. For large problem sizes (i.e. number of variables), ROL's trust region method and conjugate gradient methods exhibit good scalability for unconstrained problems. ROL handles equality constraints natively but converts inequality constraints into equality constraints with bounded slack variables. This has might degrade convergence for problems involving large number of inequality constraints. ROL has traditionally been applied to problems with analytic gradients (and Hessians)but can can be used with Dakota-provided finite-differencing approximations to the gradient of both objective and constraints. However, a user employing these is advised to use alternative optimizers such as DOT until performance of ROL improves in future releases.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Calibration](#) (when [calibration_terms](#) are specified)

7.2.28.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [rol](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.28.2 variable_tolerance

- [Keywords Area](#)
- [method](#)
- [rol](#)
- [variable_tolerance](#)

Step length-based stopping criteria for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 0.01

Description

The `variable_tolerance` keyword defines the minimum step length allowed by the optimizer and is used to determine convergence. It is applicable to `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`.

Default Behavior

The default value varies according to method as follows:

- `asynch_pattern_search`: 1.0e-2
- `coliny_cobyala`: 1.0e-4
- `coliny_pattern_search`: 1.0e-5
- `coliny_solis_wets`: 1.0e-6
- `mesh_adaptive_search`: 1.0e-6

Usage Tips

It is recommended that `variable_tolerance` be set to a value for which changes of that scale in parameter values cause negligible changes in the objective function.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    contraction_factor = 0.25
    variable_tolerance = 1.e-4
    solution_target = 1.e-6
    max_function_evaluations 500
    constraint_tolerance 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    variable_tolerance = 0.01
    seed = 1234
```

7.2.28.3 `gradient_tolerance`

- [Keywords Area](#)
- [method](#)
- [rol](#)
- [gradient_tolerance](#)

Stopping critiera based on L2 norm of gradient

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `gradient_tolerance` control defines the threshold value on the L2 norm of the objective function gradient (for unconstrained minimum or no active constraints) or the Lagrangian gradient (for constrained minimum) that indicates convergence to a stationary point. The `gradient_tolerance` control is currently defined only for the OPT++ and ROL libraries.

7.2.28.4 constraint_tolerance

- [Keywords Area](#)
- [method](#)
- [rol](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.28.5 options_file

- [Keywords Area](#)
- [method](#)
- [rol](#)
- [options_file](#)

File containing advanced ROL options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

Allow power users to override default ROL options

Default Behavior No advanced options used.

Usage Tips The `options_file` offers a final override over other options settings, which are applied with the following precedence:

1. ROL library defaults
2. Dakota hard-coded defaults
3. Settings from Dakota input file
4. Settings from the advanced `options_file`

7.2.28.6 scaling

- [Keywords Area](#)
- [method](#)
- [rol](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are

performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.28.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [rol](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.29 asynch_pattern_search

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)

Pattern search, derivative free optimization method

Topics

This keyword is related to the topics:

- [package_hopspack](#)
- [global_optimization_methods](#)

Specification

Alias: coliny_apps

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_delta	Initial step size for derivative-free optimizers
	Optional		contraction_factor	Amount by which step length is rescaled
	Optional		variable_tolerance	Step length-based stopping criteria for derivative-free optimizers
	Optional		solution_target	Stopping criteria based on objective function value
	Optional		synchronization	Select how Dakota schedules a batch of concurrent function evaluations in a parallel algorithm
	Optional		merit_function	Balance goals of reducing objective function and satisfying constraints
	Optional		constraint_penalty	Multiplier for the penalty function
	Optional		smoothing_factor	Smoothing value for smoothed penalty functions
	Optional		constraint_tolerance	Maximum allowable constraint violation still considered feasible
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers

	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The asynchronous parallel pattern search (APPS) algorithm [38] is a fully asynchronous pattern search technique in that the search along each offset direction continues without waiting for searches along other directions to finish.

Currently, APPS only supports coordinate bases with a total of $2n$ function evaluations in the pattern, and these patterns may only contract.

Concurrency

APPS exploits parallelism through the use of Dakota's concurrent function evaluations. The variant of the algorithm that is currently exposed, however, limits the amount of concurrency that can be exploited. In particular, APPS can leverage an evaluation concurrency level of at most twice the number of variables. More options that allow for greater evaluation concurrency may be exposed in future releases.

Algorithm Behavior

- `initial_delta`: the initial step length, must be positive
- `variable_tolerance`: step length used to determine convergence, must be greater than or equal to $4.4\text{e-}16$
- `contraction_factor`: amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1

Merit Functions

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-based subproblems. There are several exact and smoothed exact penalty functions that can be specified with the `merit_function` control. The options are as follows:

- `merit_max`: based on l_∞ norm
- `merit_max_smooth`: based on smoothed l_∞ norm
- `merit1`: based on l_1 norm
- `merit1_smooth`: based on smoothed l_1 norm
- `merit2`: based on l_2 norm
- `merit2_smooth`: based on smoothed l_2 norm
- `merit2_squared`: based on l_2^2 norm

The user can also specify the following to affect the merit functions:

- `constraint_penalty`
- `smoothing_parameter`

Method Independent Controls

The only method independent controls that are currently mapped to APPS are:

- [max_function_evaluations](#)
- [constraint_tolerance](#)
- [output](#)

Note that while APPS treats the constraint tolerance separately for linear and nonlinear constraints, we apply the same value to both if the user specifies `constraint_tolerance`.

The APPS internal display level is mapped to the Dakota `output` settings as follows:

- `debug`: display final solution, all input parameters, variable and constraint info, trial points, search directions, and execution details
- `verbose`: display final solution, all input parameters, variable and constraint info, and trial points
- `normal`: display final solution, all input parameters, variable and constraint summaries, and new best points
- `quiet`: display final solution and all input parameters
- `silent`: display final solution

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

7.2.29.1 `initial_delta`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [initial_delta](#)

Initial step size for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

The `initial_delta` keyword defines the size of the first search step in derivative-free optimization methods, specifically `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`. It is applied in an absolute sense to all search directions.

Default Behavior

The default value is 1.0.

Usage Tips

It is recommended that `initial_delta` be the approximate distance from the initial point to the solution. If this distance is unknown, it is advisable to err on the side of choosing an `initial_delta` that is too large or to not specify it. Relative application of `initial_delta` is not available unless the user scales the problem accordingly.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    initial_delta = .5
    contraction_factor = 0.25
    merit_function merit1_smooth
    smoothing_factor = 1.0
    constraint_tolerance = 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    initial_delta = 2.0
    seed = 1234
```

7.2.29.2 contraction_factor

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [contraction_factor](#)

Amount by which step length is rescaled

Specification

Alias: none

Argument(s): REAL

Default: 0.5

Description

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction_factor}$

7.2.29.3 `variable_tolerance`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [variable_tolerance](#)

Step length-based stopping criteria for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 0.01

Description

The `variable_tolerance` keyword defines the minimum step length allowed by the optimizer and is used to determine convergence. It is applicable to `asynch_pattern_search`, `coliny_cobyla`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`.

Default Behavior

The default value varies according to method as follows:

- `asynch_pattern_search`: 1.0e-2
- `coliny_cobyla`: 1.0e-4
- `coliny_pattern_search`: 1.0e-5
- `coliny_solis_wets`: 1.0e-6
- `mesh_adaptive_search`: 1.0e-6

Usage Tips

It is recommended that `variable_tolerance` be set to a value for which changes of that scale in parameter values cause negligible changes in the objective function.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    contraction_factor = 0.25
    variable_tolerance = 1.e-4
    solution_target = 1.e-6
    max_function_evaluations 500
    constraint_tolerance 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    variable_tolerance = 0.01
    seed = 1234
```

7.2.29.4 `solution_target`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: no target

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.29.5 `synchronization`

- [Keywords Area](#)
- [method](#)

- [asynch_pattern_search](#)
- [synchronization](#)

Select how Dakota schedules a batch of concurrent function evaluations in a parallel algorithm

Specification

Alias: none

Argument(s): none

Default: method specific

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Synchronization (Group 1)	Dakota Keyword blocking	Dakota Keyword Description Complete concurrent evaluation of all evaluations within a batch
			nonblocking	A batch of concurrent evaluations are queried for completions and a partial set is returned to the algorithm

Description

The `synchronization` specification can be used to specify the use of either `blocking` or `nonblocking` schedulers, where the former must block until all evaluations are completed and the latter does not block, returning only the partial set that has been completed for a particular call to the scheduler.

7.2.29.5.1 `blocking`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [synchronization](#)
- [blocking](#)

Complete concurrent evaluation of all evaluations within a batch

Specification

Alias: none

Argument(s): none

Description

In the `blocking` case, all of the concurrent evaluations in the batch must be completed and all results are returned together to the algorithm.

These runs are reproducible, assuming use of the same seed in cases with random number generation.

7.2.29.5.2 `nonblocking`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [synchronization](#)
- [nonblocking](#)

A batch of concurrent evaluations are queried for completions and a partial set is returned to the algorithm

Specification

Alias: none

Argument(s): none

Description

In the `nonblocking` case, the set of running evaluations is queried for completion and a partial set of results is returned to an asynchronous algorithm, which can then generate new evaluations based on these partial results.

In all cases, the algorithm steps will be subject to parallel timing variabilities / race conditions such that these runs will not generally be repeatable regardless of seed specification.

7.2.29.6 `merit_function`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none

Argument(s): none

Default: `merit2_squared`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
			merit_max	Nonsmoothed merit function
	Required (<i>Choose One</i>)	Merit Function (Group 1)	merit_max_smooth	Smoothed merit function
			merit1	Nonsmoothed merit function
			merit1_smooth	Smoothed merit function
			merit2	Nonsmoothed merit function
			merit2_smooth	Smoothed merit function
			merit2_squared	Nonsmoothed merit function

Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

7.2.29.6.1 `merit_max`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)
- [merit_max](#)

Nonsmoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

`merit_max`: based on l_∞ norm

7.2.29.6.2 `merit_max_smooth`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)
- [merit_max_smooth](#)

Smoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

`merit_max_smooth`: based on smoothed l_∞ norm

7.2.29.6.3 merit

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)
- [merit1](#)

Nonsmoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

- `merit1`: based on l_1 norm

7.2.29.6.4 merit1_smooth

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)
- [merit1_smooth](#)

Smoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

`merit1_smooth`: based on smoothed ℓ_1 norm

7.2.29.6.5 merit2

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)
- [merit2](#)

Nonsmoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

`merit2`: based on ℓ_2 norm

7.2.29.6.6 merit2_smooth

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)
- [merit2_smooth](#)

Smoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

`merit2_smooth`: based on smoothed ℓ_2 norm

7.2.29.6.7 merit2_squared

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [merit_function](#)
- [merit2_squared](#)

Nonsmoothed merit function

Specification

Alias: none

Argument(s): none

Description

APPS solves nonlinearly constrained problems by solving a sequence of linearly constrained merit function-base subproblems. There are several exact and smoothed exact penalty functions.

`merit2_squared`: based on ℓ_2^2 norm

7.2.29.7 constraint_penalty

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [constraint_penalty](#)

Multiplier for the penalty function

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

7.2.29.8 `smoothing_factor`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [smoothing_factor](#)

Smoothing value for smoothed penalty functions

Specification

Alias: none

Argument(s): REAL

Default: 0.0

Description

- `smoothing_parameter`: initial smoothing value for smoothed penalty functions, must be between 0 and 1 (inclusive)

7.2.29.9 `constraint_tolerance`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [constraint_tolerance](#)

Maximum allowable constraint violation still considered feasible

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: Library default

Description

Sets the maximum allowable value of infeasibility that any constraint in an optimization problem may possess and still be considered to be satisfied.

If a constraint's violation is greater than this value then it is considered to be violated by the optimization algorithm. This specification gives some control over how tightly the constraints will be satisfied at convergence of the algorithm. However, if the value is set too small the algorithm may terminate with one or more constraints being violated.

Defaults

Defaults vary depending on the method.

- CONMIN optimizers: 1.0e-3
- DOT constrained optimizers: 3.0e-3
- [surrogate_based_local](#) : 1.0e-4

7.2.29.10 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.29.11 scaling

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto`, `log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.29.12 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [asynch_pattern_search](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.30 mesh_adaptive_search

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)

Finds optimal variable values using adaptive mesh-based search

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_delta	Initial step size for derivative-free optimizers
	Optional		variable_tolerance	Step length-based stopping criteria for derivative-free optimizers
	Optional		function_precision	Specify the maximum precision of the analysis code responses
	Optional		seed	Seed of the random number generator
	Optional		history_file	Name of file where mesh adaptive search records all evaluation points.

	Optional	display_format	Information to be reported from mesh adaptive search's internal records.
	Optional	variable_-neighborhood_-search	Percentage of evaluations to do to escape local minima.
	Optional	neighbor_order	Number of dimensions in which to perturb categorical variables.
	Optional	display_all_-evaluations	Shows mesh adaptive search's internally held list of all evaluations
	Optional	use_surrogate	Surrogate model usage mode for mesh adaptive search
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	max_function_-evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The mesh adaptive direct search algorithm [8] is a derivative-free generalized pattern search in which the set of points evaluated becomes increasingly dense, leading to good convergence properties. It can handle unconstrained problems as well as those with bound constraints and general nonlinear constraints. Furthermore, it can handle continuous, discrete, and categorical variables.

Default Behavior

By default, `mesh_adaptive_search` operates on design variables. The types of variables can be expanded through the use of the `active` keyword in the `variables` block in the Dakota input file. Categorical variables, however, must be limited to design variables.

Expected Outputs

The best objective function value achieved and associated parameter and constraint values can be found at the end of the Dakota output. The method's internally summarized iteration history will appear in the screen output by default, with the option to control the method's output through Dakota's output level. It also generates a history file containing a list of all function evaluations done.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

Additional Discussion

The mesh adaptive direct search method is made available in Dakota through the NOMAD software [2], available to the public under the GNU LGPL from <http://www.gerad.ca/nomad>.

Examples

The following is an example of a Dakota input file that makes use of `mesh_adaptive_search` to optimize the textbook function.

```
method,
  mesh_adaptive_search
  seed = 1234

variables,
  continuous_design = 3
  initial_point    -1.0   1.5   2.0
  upper_bounds     10.0  10.0  10.0
  lower_bounds     -10.0 -10.0 -10.0
  descriptors      'x1'  'x2'  'x3'

interface,
  direct
  analysis_driver = 'text_book'

responses,
  objective_functions = 1
  no_gradients
  no_hessians
```

The best function value and associated parameters are found at the end of the Dakota output.

```
<<<< Function evaluation summary: 674 total (674 new, 0 duplicate)
<<<< Best parameters      =
                        1.0000000000e+00 x1
                        1.0000000000e+00 x2
                        1.0000000000e+00 x3
<<<< Best objective function =
                        1.0735377280e-52
<<<< Best evaluation ID: 658
```

A NOMAD-generated iteration summary is also printed to the screen.

```
MADS run {
```

BBE OBJ

```
1 17.0625000000
2 1.0625000000
13 0.0625000000
24 0.0002441406
41 0.0000314713
43 0.0000028610
54 0.0000000037
83 0.0000000000
105 0.0000000000
112 0.0000000000
114 0.0000000000
135 0.0000000000
142 0.0000000000
153 0.0000000000
159 0.0000000000
171 0.0000000000
193 0.0000000000
200 0.0000000000
207 0.0000000000
223 0.0000000000
229 0.0000000000
250 0.0000000000
266 0.0000000000
282 0.0000000000
288 0.0000000000
314 0.0000000000
320 0.0000000000
321 0.0000000000
327 0.0000000000
354 0.0000000000
361 0.0000000000
372 0.0000000000
373 0.0000000000
389 0.0000000000
400 0.0000000000
417 0.0000000000
444 0.0000000000
459 0.0000000000
461 0.0000000000
488 0.0000000000
492 0.0000000000
494 0.0000000000
501 0.0000000000
518 0.0000000000
530 0.0000000000
537 0.0000000000
564 0.0000000000
566 0.0000000000
583 0.0000000000
590 0.0000000000
592 0.0000000000
604 0.0000000000
606 0.0000000000
629 0.0000000000
636 0.0000000000
658 0.0000000000
674 0.0000000000
```

} end of run (mesh size reached NOMAD precision)

```
blackbox evaluations : 674
best feasible solution : ( 1 1 1 ) h=0 f=1.073537728e-52
```

7.2.30.1 initial_delta

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)

- [initial_delta](#)

Initial step size for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

The `initial_delta` keyword defines the size of the first search step in derivative-free optimization methods, specifically `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`. It is applied in an absolute sense to all search directions.

Default Behavior

The default value is 1.0.

Usage Tips

It is recommended that `initial_delta` be the approximate distance from the initial point to the solution. If this distance is unknown, it is advisable to err on the side of choosing an `initial_delta` that is too large or to not specify it. Relative application of `initial_delta` is not available unless the user scales the problem accordingly.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    initial_delta = .5
    contraction_factor = 0.25
    merit_function merit1_smooth
    smoothing_factor = 1.0
    constraint_tolerance = 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    initial_delta = 2.0
    seed = 1234
```

7.2.30.2 variable_tolerance

- [Keywords Area](#)

- [method](#)
- [mesh_adaptive_search](#)
- [variable_tolerance](#)

Step length-based stopping criteria for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-6

Description

The `variable_tolerance` keyword defines the minimum step length allowed by the optimizer and is used to determine convergence. It is applicable to `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`.

Default Behavior

The default value varies according to method as follows:

- `asynch_pattern_search`: 1.0e-2
- `coliny_cobyala`: 1.0e-4
- `coliny_pattern_search`: 1.0e-5
- `coliny_solis_wets`: 1.0e-6
- `mesh_adaptive_search`: 1.0e-6

Usage Tips

It is recommended that `variable_tolerance` be set to a value for which changes of that scale in parameter values cause negligible changes in the objective function.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    contraction_factor = 0.25
    variable_tolerance = 1.e-4
    solution_target = 1.e-6
    max_function_evaluations 500
    constraint_tolerance 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    variable_tolerance = 0.01
    seed = 1234
```

7.2.30.3 `function_precision`

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [function_precision](#)

Specify the maximum precision of the analysis code responses

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

The `function_precision` control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.

7.2.30.4 `seed`

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.30.5 history_file

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [history_file](#)

Name of file where mesh adaptive search records all evaluation points.

Specification

Alias: none

Argument(s): STRING

Default: mads_history

Description

The `history_file` is used to specify the name of a file to which mesh adaptive direct search will write its own list of evaluated points.

Default Behavior

By default, mesh adaptive direct search will write the list of evaluation points in a file named `mads_history.xxxx`, where `xxxx` corresponds to a randomly generated number. Dakota's output level controls the method's level of output to file.

Examples

The example below shows the syntax for specifying the name of the history file.

```
method
  mesh_adaptive_search
    history_file = 'output.log'
    seed = 1234
```

7.2.30.6 display_format

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [display_format](#)

Information to be reported from mesh adaptive search's internal records.

Specification

Alias: none

Argument(s): STRING

Description

The `display_format` keyword is used to specify the set of information to be reported by the mesh adaptive direct search method. This is information mostly internal to the method and not reported via Dakota output.

Default Behavior

By default, only the number of function evaluations (bbe) and the objective function value (obj) are reported.

The full list of options is as follows. Note that case does not matter.

- BBE: Blackbox evaluations.
- BBO: Blackbox outputs.
- EVAL: Evaluations (includes cache hits).
- MESH_INDEX: Mesh index.
- MESH_SIZE: Mesh size parameter.
- OBJ: Objective function value.
- POLL_SIZE: Poll size parameter.
- SOL: Solution, with format iSOLj where i and j are two (optional) strings: i will be displayed before each coordinate, and j after each coordinate (except the last).
- STAT_AVG: The AVG statistic.
- STAT_SUM: The SUM statistic defined by argument.
- TIME: Wall-clock time.
- VARi: Value of variable i. The index 0 corresponds to the first variable.

Expected Outputs

A list of the requested information will be printed to the screen.

Usage Tips

This will most likely only be useful for power users who want to understand and/or report more detailed information on method behavior.

Examples

The following example shows the syntax for specifying `display_format`. Note that all desired information options should be listed within a single string.

```
method
  mesh_adaptive_search
  display_format 'bbe obj poll_size'
  seed = 1234
```

Below is the output reported for the above example.

```
MADS run {
  BBE OBJ POLL_SIZE

    1  17.0625000000  2.0000000000 2.0000000000 2.0000000000
    2  1.0625000000  2.0000000000 2.0000000000 2.0000000000
   13  0.0625000000  1.0000000000 1.0000000000 1.0000000000
   24  0.0002441406  0.5000000000 0.5000000000 0.5000000000
   41  0.0000314713  0.1250000000 0.1250000000 0.1250000000
   43  0.0000028610  0.2500000000 0.2500000000 0.2500000000
   54  0.0000000037  0.1250000000 0.1250000000 0.1250000000
   83  0.0000000000  0.0078125000 0.0078125000 0.0078125000
  105  0.0000000000  0.0009765625 0.0009765625 0.0009765625
  112  0.0000000000  0.0009765625 0.0009765625 0.0009765625
  114  0.0000000000  0.0019531250 0.0019531250 0.0019531250
  135  0.0000000000  0.0004882812 0.0004882812 0.0004882812
  142  0.0000000000  0.0004882812 0.0004882812 0.0004882812
  153  0.0000000000  0.0004882812 0.0004882812 0.0004882812
  159  0.0000000000  0.0009765625 0.0009765625 0.0009765625
  171  0.0000000000  0.0004882812 0.0004882812 0.0004882812
  193  0.0000000000  0.0000610352 0.0000610352 0.0000610352
  200  0.0000000000  0.0000610352 0.0000610352 0.0000610352
  207  0.0000000000  0.0000610352 0.0000610352 0.0000610352
  223  0.0000000000  0.0000305176 0.0000305176 0.0000305176
  229  0.0000000000  0.0000610352 0.0000610352 0.0000610352
  250  0.0000000000  0.0000152588 0.0000152588 0.0000152588
  266  0.0000000000  0.0000076294 0.0000076294 0.0000076294
  282  0.0000000000  0.0000038147 0.0000038147 0.0000038147
  288  0.0000000000  0.0000076294 0.0000076294 0.0000076294
  314  0.0000000000  0.0000009537 0.0000009537 0.0000009537
  320  0.0000000000  0.0000019073 0.0000019073 0.0000019073
  321  0.0000000000  0.0000038147 0.0000038147 0.0000038147
  327  0.0000000000  0.0000076294 0.0000076294 0.0000076294
  354  0.0000000000  0.0000004768 0.0000004768 0.0000004768
  361  0.0000000000  0.0000004768 0.0000004768 0.0000004768
  372  0.0000000000  0.0000004768 0.0000004768 0.0000004768
  373  0.0000000000  0.0000009537 0.0000009537 0.0000009537
  389  0.0000000000  0.0000004768 0.0000004768 0.0000004768
  400  0.0000000000  0.0000004768 0.0000004768 0.0000004768
  417  0.0000000000  0.0000001192 0.0000001192 0.0000001192
  444  0.0000000000  0.0000000075 0.0000000075 0.0000000075
  459  0.0000000000  0.0000000037 0.0000000037 0.0000000037
  461  0.0000000000  0.0000000075 0.0000000075 0.0000000075
  488  0.0000000000  0.0000000005 0.0000000005 0.0000000005
  492  0.0000000000  0.0000000009 0.0000000009 0.0000000009
  494  0.0000000000  0.0000000019 0.0000000019 0.0000000019
  501  0.0000000000  0.0000000019 0.0000000019 0.0000000019
  518  0.0000000000  0.0000000005 0.0000000005 0.0000000005
```

```

530 0.0000000000 0.0000000002 0.0000000002 0.0000000002
537 0.0000000000 0.0000000002 0.0000000002 0.0000000002
564 0.0000000000 0.0000000000 0.0000000000 0.0000000000
566 0.0000000000 0.0000000000 0.0000000000 0.0000000000
583 0.0000000000 0.0000000000 0.0000000000 0.0000000000
590 0.0000000000 0.0000000000 0.0000000000 0.0000000000
592 0.0000000000 0.0000000000 0.0000000000 0.0000000000
604 0.0000000000 0.0000000000 0.0000000000 0.0000000000
606 0.0000000000 0.0000000000 0.0000000000 0.0000000000
629 0.0000000000 0.0000000000 0.0000000000 0.0000000000
636 0.0000000000 0.0000000000 0.0000000000 0.0000000000
658 0.0000000000 0.0000000000 0.0000000000 0.0000000000
674 0.0000000000 0.0000000000 0.0000000000 0.0000000000

```

```
} end of run (mesh size reached NOMAD precision)
```

```

blackbox evaluations      : 674
best feasible solution   : ( 1 1 1 ) h=0 f=1.073537728e-52

```

See Also

These keywords may also be of interest:

- [display_all_evaluations](#)

7.2.30.7 variable_neighborhood_search

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [variable_neighborhood_search](#)

Percentage of evaluations to do to escape local minima.

Specification

Alias: none

Argument(s): REAL

Default: 0.0

Description

The `variable_neighborhood_search` keyword is used to set the percentage (in decimal form) of function evaluations used to escape local minima. The mesh adaptive direct search method will try to perform a maximum of that percentage of the function evaluations within this more extensive search.

Default Behavior

By default, `variable_neighborhood_search` is not used.

Usage Tips

Using `variable_neighborhood_search` results in an increased number of function evaluations. If the desired result is a local minimum, the added cost is of little or no value, so the recommendation is not to use it. If the desired result is the best local minimum possible within a computational budget, then there is value in setting this parameter. Note that the higher the value, the greater the computational cost. The value should be no greater than 1.0.

Examples

The following example shows the syntax used to set `variable_neighborhood_search`.

```
method
  mesh_adaptive_search
    seed = 1234
    variable_neighborhood_search = 0.1
```

7.2.30.8 neighbor_order

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [neighbor_order](#)

Number of dimensions in which to perturb categorical variables.

Specification

Alias: none

Argument(s): INTEGER

Description

The `neighbor_order` keyword allows the user to specify the number of categorical dimensions to perturb when determining neighboring points that will be used by the mesh adaptive direct search method to augment its search. When greater than 1, the neighbors are defined from the tensor product of the admissible 1-dimensional perturbations.

Default Behavior

By default, the categorical neighbors will be defined by perturbing only one categorical variable at a time (according to the corresponding `adjacency_matrix`; see [adjacency_matrix](#)) while leaving the others fixed at their current values.

Usage Tips

The maximum meaningful value `neighbor_order` can take on is the number of categorical variables.

Examples

In this example, suppose we have the following categorical variables and associated adjacency matrices.

```
variables
  discrete_design_set
    real = 2
    categorical 'yes' 'yes'
    num_set_values = 3 5
    set_values = 1.2 2.3 3.4
                1.2 3.3 4.4 5.5 7.7
    adjacency_matrix = 1 1 0
                      1 1 1
                      0 1 1
                      1 0 1 0 1
                      0 1 0 1 0
                      1 0 1 0 1
                      0 1 0 1 0
                      1 0 1 0 1
```

Also suppose that we have the following method specification.

```
method
  mesh_adaptive_search
    seed = 1234
```

If the mesh adaptive direct search is at the point (1.2, 1.2), then the neighbors will be defined by the default 1-dimensional perturbations and would be the following:

```
(2.3, 1.2)
(1.2, 4.4)
(1.2, 7.7)
```

If, instead, the method specification is the following:

```
method
  mesh_adaptive_search
    seed = 1234
    neighbor_order = 2
```

The neighbors will be defined by 2-dimensional perturbations defined from the tensor product of the 1-dimensional perturbation and would be the following:

```
(2.3, 1.2)
(2.3, 4.4)
(2.3, 7.7)
(1.2, 4.4)
(1.2, 7.7)
```

See Also

These keywords may also be of interest:

- [adjacency_matrix](#)

7.2.30.9 display_all_evaluations

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [display_all_evaluations](#)

Shows mesh adaptive search's internally held list of all evaluations

Specification

Alias: none

Argument(s): none

Default: false

Description

If set, `display_all_evaluations` will instruct the mesh adaptive direct search method to print out its own record of all evaluations. The information reported may be controlled using `display_format`.

Default Behavior

By default, mesh adaptive direct search does not report information on all evaluations, only on those for which an improvement in the objective function is found.

Expected Outputs

The information specified by `display_format` will be reported to the screen for every function evaluation.

Usage Tips

This will most likely only be useful for power users who want to understand and/or report more detailed information on method behavior.

Examples

The following example shows the syntax for specifying `display_all_evaluations`.

```
method
  mesh_adaptive_search
  display_all_evaluations
  max_function evaluations=20
  seed = 1234
```

Note that the output below reports information (default for `display_format`) for all function evaluations.

```
MADS run {
  BBE OBJ
  1 17.0625000000
  2 1.0625000000
  3 1297.0625000000
  4 257.0625000000
  5 81.0625000000
  6 151.0625000000
  7 1051.0625000000
  8 40.0625000000
  9 17.0625000000
  10 40.0625000000
  11 1.0625000000
  12 102.0625000000
  13 0.0625000000
  14 231.0625000000
  15 16.0625000000
  16 5.0625000000
  17 16.0625000000
  18 71.0625000000
  19 0.0625000000
  20 1.0625000000
} end of run (max number of blackbox evaluations)
blackbox evaluations : 20
best feasible solution : ( 1 0.5 1 ) h=0 f=0.0625
```

That is in contrast with what would be reported by default.

```
MADS run {
  BBE OBJ
```

```

1 17.0625000000
2 1.0625000000
13 0.0625000000
20 0.0625000000

} end of run (max number of blackbox evaluations)

blackbox evaluations          : 20
best feasible solution      : ( 1 0.5 1 ) h=0 f=0.0625

```

See Also

These keywords may also be of interest:

- [display_format](#)

7.2.30.10 use_surrogate

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [use_surrogate](#)

Surrogate model usage mode for mesh adaptive search

Specification

Alias: none

Argument(s): none

Default: optimize

Child Keywords:

	Required/Optional <i>Required(Choose One)</i>	Description of Group Surrogate Purpose (Group 1)	Dakota Keyword inform_search	Dakota Keyword Description Surrogate informs evaluation order in mesh adaptive search
			optimize	Surrogate is used in lieu of true model for mesh adaptive search

Description

The `use_surrogate` keyword is used to define how a surrogate model (if one is provided) is to be used by `mesh_adaptive_search`. There are two approaches available: `inform_search` uses the surrogate to sort list of trial points and subsequently the true function is evaluated on the most promising points first. Both true function and surrogate are used interchangeably within the method. `optimize` forces the use of a surrogate in lieu of the true model and thus the true function is never invoked except to construct the surrogate.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Default Behavior

By default, `mesh_adaptive_search` follows behaviour provided by `optimize` option.

Examples

The following example shows the syntax used to set `use_surrogate`.

```
method,
  mesh_adaptive_search
  model_pointer = 'SURROGATE'
  use_surrogate inform_search

model,
  id_model = 'SURROGATE'
  surrogate global
  polynomial quadratic
  dace_method_pointer = 'SAMPLING'

variables,
  continuous_design = 3
  initial_point    -1.0    1.5    2.0
  upper_bounds     10.0   10.0   10.0
  lower_bounds    -10.0  -10.0 -10.0
  descriptors      'x1'   'x2'   'x3'
  discrete_design_range = 2
  initial_point     2      2
  lower_bounds      1      1
  upper_bounds      4      9
  descriptors       'y1'   'y2'
  discrete_design_set
    real = 2
    elements_per_variable = 4 5
    elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
    descriptors      'y3'   'y4'
    integer = 2
    elements_per_variable = 2 2
    elements = 4 7 8 9
    descriptors      'z1'   'z2'

method,
  id_method = 'SAMPLING'
  model_pointer = 'TRUTH'
  sampling
  samples = 55

model,
  id_model = 'TRUTH'
  single
  interface_pointer = 'TRUE_FN'

interface,
  id_interface = 'TRUE_FN'
  direct
  analysis_driver = 'text_book'

responses,
  objective_functions = 1
  no_gradients
  no_hessians
```

7.2.30.10.1 inform_search

- [Keywords Area](#)

- [method](#)
- [mesh_adaptive_search](#)
- [use_surrogate](#)
- [inform_search](#)

Surrogate informs evaluation order in mesh adaptive search

Specification

Alias: none

Argument(s): none

Description

When `inform_search` is specified with `use_surrogate`, `mesh_adaptive_search` uses the surrogate to sort list of trial points and subsequently the true function is evaluated on the most promising points first. Both true function and surrogate are used interchangeably within the method.

Default Behavior

`inform_search` is not the default surrogate usage mode.

Expected Output

The user can expect to see both the number of true model evaluations and the number of approximation (i.e., surrogate) evaluations reported in the Dakota screen output. The former captures the sum of truth evaluations done for the surrogate construction and for the optimization.

Usage Tips

When `inform_search` is specified, the `maximum_function_evaluations` keyword applies to only the optimization method and does not account for evaluations needed to construct the surrogate. If the user has a strict evaluation budget, they should set `maximum_function_evaluations` such that `evaluation budget = number of evaluations to construct surrogate + maximum_function_evaluations`.

Examples

The following example shows the syntax used to set `use_surrogate` to optimize.

```
method,
  mesh_adaptive_search
  model_pointer = 'SURROGATE'
  use_surrogate inform_search

model,
  id_model = 'SURROGATE'
  surrogate global
  polynomial quadratic
  dace_method_pointer = 'SAMPLING'

variables,
  continuous_design = 3
  initial_point      -1.0   1.5   2.0
  upper_bounds       10.0  10.0  10.0
  lower_bounds       -10.0 -10.0 -10.0
  descriptors         'x1'  'x2'  'x3'
  discrete_design_range = 2
  initial_point      2     2
  lower_bounds       1     1
  upper_bounds       4     9
```

```

descriptors      'y1'   'y2'
discrete_design_set
  real = 2
  elements_per_variable = 4 5
  elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
  descriptors      'y3'   'y4'
integer = 2
  elements_per_variable = 2 2
  elements = 4 7 8 9
  descriptors      'z1'   'z2'

method,
  id_method = 'SAMPLING'
  model_pointer = 'TRUTH'
  sampling
    samples = 55

model,
  id_model = 'TRUTH'
  single
    interface_pointer = 'TRUE_FN'

interface,
  id_interface = 'TRUE_FN'
  direct
    analysis_driver = 'text_book'

responses,
  objective_functions = 1
  no_gradients
  no_hessians

```

The following will appear toward the end of the screen output when Dakota is run on this example. The number of true function evaluations includes the 55 evaluations that were done to construct the surrogate (as specified in the SAMPLING method block) plus the number of truth evaluations done by `mesh_adaptive_search`.

```

<<<<< Function evaluation summary (APPROX_INTERFACE): 1660 total (1660 new, 0 duplicate)
<<<<< Function evaluation summary (TRUE_FN): 795 total (795 new, 0 duplicate)

```

7.2.30.10.2 optimize

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [use_surrogate](#)
- [optimize](#)

Surrogate is used in lieu of true model for mesh adaptive search

Specification

Alias: none

Argument(s): none

Description

When `optimize` is specified with `use_surrogate`, `mesh_adaptive_search` will use the surrogate in lieu of the true model and thus the true function is never invoked except to construct the surrogate.

Default Behavior

optimize is the default surrogate usage mode.

Expected Output

The user can expect to see both the number of true model evaluations and the number of approximation (i.e., surrogate) evaluations reported in the Dakota screen output. The former should equal the number of truth evaluations done for the surrogate construction.

Examples

The following example shows the syntax used to set `use_surrogate` to `optimize`.

```
method,
  mesh_adaptive_search
  model_pointer = 'SURROGATE'
  use_surrogate optimize

model,
  id_model = 'SURROGATE'
  surrogate global
  polynomial quadratic
  dace_method_pointer = 'SAMPLING'

variables,
  continuous_design = 3
  initial_point    -1.0    1.5    2.0
  upper_bounds     10.0    10.0   10.0
  lower_bounds     -10.0   -10.0  -10.0
  descriptors      'x1'    'x2'   'x3'
  discrete_design_range = 2
  initial_point    2      2
  lower_bounds     1      1
  upper_bounds     4      9
  descriptors      'y1'    'y2'
  discrete_design_set
    real = 2
    elements_per_variable = 4 5
    elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
    descriptors    'y3'    'y4'
  integer = 2
  elements_per_variable = 2 2
  elements = 4 7 8 9
  descriptors      'z1'    'z2'

method,
  id_method = 'SAMPLING'
  model_pointer = 'TRUTH'
  sampling
  samples = 55

model,
  id_model = 'TRUTH'
  single
  interface_pointer = 'TRUE_FN'

interface,
  id_interface = 'TRUE_FN'
  direct
  analysis_driver = 'text_book'

responses,
  objective_functions = 1
  no_gradients
  no_hessians
```

The following will appear toward the end of the screen output when Dakota is run on this example. The number of true function evaluations includes only the 55 evaluations that were done to construct the surrogate (as specified in the `SAMPLING` method block).

```
<<<<< Function evaluation summary (APPROX_INTERFACE): 221 total (221 new, 0 duplicate)
```

```
<<<<< Function evaluation summary (TRUE_FN): 55 total (55 new, 0 duplicate)
```

7.2.30.11 max_iterations

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.30.12 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.30.13 `scaling`

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.30.14 model_pointer

- [Keywords Area](#)
- [method](#)
- [mesh_adaptive_search](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.31 nowpac

- [Keywords Area](#)
- [method](#)
- [nowpac](#)

Gradient-free inequality-constrained optimization using Nonlinear Optimization With Path Augmented Constraints (NOWPAC).

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		trust_region	Use trust region as the globalization strategy.
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

NOWPAC is a provably-convergent gradient-free optimization method from MIT that solves a series of trust region surrogate-based subproblems to generate improving steps. Due to its use of an interior penalty scheme and enforcement of strict feasibility, it does not support linear or nonlinear *equality* constraints. As opposed to the stochastic version (SNOWPAC), NOWPAC does not currently support a feasibility restoration mode, so it is necessary to start from a feasible design.

Note: (S)NOWPAC is not configured with Dakota by default and requires a separate installation of the NOWPAC distribution from MIT, combined with its TPLs of Eigen and NLOPT.

Examples

```
method
  nowpac
    max_function_evaluations = 1000
    convergence_tolerance = 1e-4
    trust_region
      initial_size = 0.10
      minimum_size = 1.0e-6
      contract_threshold = 0.25
      expand_threshold = 0.75
      contraction_factor = 0.50
      expansion_factor = 1.50
```

7.2.31.1 trust_region

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [trust_region](#)

Use trust region as the globalization strategy.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_size	Trust region initial size (relative to bounds)
	Optional		minimum_size	Trust region minimum size
	Optional		contract_threshold	Shrink trust region if trust region ratio is below this value
	Optional		expand_threshold	Expand trust region if trust region ratio is above this value
	Optional		contraction_factor	Amount by which step length is rescaled
	Optional		expansion_factor	Trust region expansion factor

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.31.1.1 `initial_size`

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [trust_region](#)
- [initial_size](#)

Trust region initial size (relative to bounds)

Specification

Alias: none

Argument(s): REALLIST

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

All of these specifications are `REAL` scalars, with the exception of the specification of `initial_size`, which is a `REALLIST`. This array corresponds to the case when there are more than 2 model forms or discretizations within a model hierarchy. The default `initial_size` involves a recursive halving of the global bounds for each trust region in the hierarchy: e.g., a scalar value of `.5` for a single trust region managing two model forms/discretizations, or an array of `(.125, .25, .5)` for four model forms/discretizations (three trust regions ordered from the lowest to highest fidelity surrogate, with model four as truth).

7.2.31.1.2 `minimum_size`

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [trust_region](#)
- [minimum_size](#)

Trust region minimum size

Specification

Alias: none

Argument(s): REAL

Default: 1.e-6

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The

command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.31.1.3 `contract_threshold`

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [trust_region](#)
- [contract_threshold](#)

Shrink trust region if trust region ratio is below this value

Specification

Alias: none

Argument(s): REAL

Default: 0.25

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.31.1.4 `expand_threshold`

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [trust_region](#)
- [expand_threshold](#)

Expand trust region if trust region ratio is above this value

Specification

Alias: none

Argument(s): REAL

Default: 0.75

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.31.1.5 `contraction_factor`

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [trust_region](#)
- [contraction_factor](#)

Amount by which step length is rescaled

Specification

Alias: none

Argument(s): REAL

Default: 0.25

Description

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction_factor}$

7.2.31.1.6 `expansion_factor`

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [trust_region](#)
- [expansion_factor](#)

Trust region expansion factor

Specification

Alias: none

Argument(s): REAL

Default: 2.0

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.31.2 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.31.3 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.31.4 scaling

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.31.5 model_pointer

- [Keywords Area](#)
- [method](#)
- [nowpac](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.32 snowpac

- [Keywords Area](#)
- [method](#)
- [snowpac](#)

Stochastic version of NOWPAC that incorporates error estimates and noise mitigation.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed	Seed of the random number generator
	Optional		trust_region	Use trust region as the globalization strategy.
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

NOWPAC is a provably-convergent gradient-free optimization method from MIT that solves a series of trust region surrogate-based subproblems to generate improving steps. The stochastic version is SNOWPAC, which incorporates noise estimates in its objective and inequality constraints. SNOWPAC modifies its trust region controls and

adds smoothing from a Gaussian process surrogate in order to mitigate noise. SNOWPAC also supports a feasibility restoration mode, so it is not necessary to start from a feasible design.

Note: (S)NOWPAC is not configured with Dakota by default and requires a separate installation of the NOWPAC distribution from MIT, combined with its TPLs of Eigen and NLOPT.

Examples

Relative to the NOWPAC specification, SNOWPAC supports a seed control for repeatability of runs and also requires the return of error estimates from the underlying evaluator (e.g., UQ method such as Monte Carlo sampling).

```
method,
  snowpac
    seed = 2504
    max_function_evaluations = 1000
    convergence_tolerance = 1e-4
    trust_region
      initial_size = 0.10
      minimum_size = 1.0e-6
      contract_threshold = 0.25
      expand_threshold   = 0.75
      contraction_factor = 0.50
      expansion_factor   = 1.50
```

7.2.32.1 seed

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.32.2 trust_region

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [trust_region](#)

Use trust region as the globalization strategy.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_size	Trust region initial size (relative to bounds)
	Optional		minimum_size	Trust region minimum size
	Optional		contract_threshold	Shrink trust region if trust region ratio is below this value
	Optional		expand_threshold	Expand trust region if trust region ratio is above this value
	Optional		contraction_factor	Amount by which step length is rescaled
	Optional		expansion_factor	Trust region expansion factor

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.32.2.1 `initial_size`

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [trust_region](#)
- [initial_size](#)

Trust region initial size (relative to bounds)

Specification

Alias: none

Argument(s): REALLIST

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

All of these specifications are `REAL` scalars, with the exception of the specification of `initial_size`, which is a `REALLIST`. This array corresponds to the case when there are more than 2 model forms or discretizations within a model hierarchy. The default `initial_size` involves a recursive halving of the global bounds for each trust region in the hierarchy: e.g., a scalar value of .5 for a single trust region managing two model forms/discretizations, or an array of (.125, .25, .5) for four model forms/discretizations (three trust regions ordered from the lowest to highest fidelity surrogate, with model four as truth).

7.2.32.2.2 `minimum_size`

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [trust_region](#)
- [minimum_size](#)

Trust region minimum size

Specification

Alias: none

Argument(s): REAL

Default: 1.e-6

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.32.2.3 `contract_threshold`

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [trust_region](#)
- [contract_threshold](#)

Shrink trust region if trust region ratio is below this value

Specification

Alias: none

Argument(s): REAL

Default: 0.25

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.32.2.4 `expand_threshold`

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [trust_region](#)
- [expand_threshold](#)

Expand trust region if trust region ratio is above this value

Specification

Alias: none

Argument(s): REAL

Default: 0.75

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.32.2.5 `contraction_factor`

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [trust_region](#)
- [contraction_factor](#)

Amount by which step length is rescaled

Specification

Alias: none

Argument(s): REAL

Default: 0.25

Description

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction_factor}$

7.2.32.2.6 `expansion_factor`

- [Keywords Area](#)
- [method](#)
- [snowpac](#)

- [trust_region](#)
- [expansion_factor](#)

Trust region expansion factor

Specification

Alias: none

Argument(s): REAL

Default: 2.0

Description

The `trust_region` optional group specification can be used to specify the initial size of the trust region (using `initial_size`) relative to the total variable bounds, the minimum size of the trust region (using `minimum_size`), the contraction factor for the trust region size (using `contraction_factor`) used when the surrogate model is performing poorly, and the expansion factor for the trust region size (using `expansion_factor`) used when the the surrogate model is performing well. Two additional commands are the trust region size contraction threshold (using `contract_threshold`) and the trust region size expansion threshold (using `expand_threshold`). These two commands are related to what is called the trust region ratio, which is the actual decrease in the truth model divided by the predicted decrease in the truth model in the current trust region. The command `contract_threshold` sets the minimum acceptable value for the trust region ratio, i.e., values below this threshold cause the trust region to shrink for the next SBL iteration. The command `expand_threshold` determines the trust region value above which the trust region will expand for the next SBL iteration.

7.2.32.3 max_iterations

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.32.4 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.32.5 scaling

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. auto - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- auto, log - optional
- value - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100

responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.32.6 model_pointer

- [Keywords Area](#)
- [method](#)
- [snowpac](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```



```

variables
  uniform_uncertain = 2
  lower_bounds = 0.  0.
  upper_bounds = 1.  1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.33 moga

- [Keywords Area](#)
- [method](#)
- [moga](#)

Multi-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)

Topics

This keyword is related to the topics:

- [package_jega](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		fitness_type	Select the fitness type for JEGA methods
	Optional		replacement_type	Select a replacement type for JEGA methods
	Optional		niching_type	Specify the type of niching pressure
	Optional		convergence_type	Select the convergence type for JEGA methods
	Optional		postprocessor_type	Post process the final solution from moga

	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	population_size	Set the initial population size in JEGA methods
	Optional	log_file	Specify the name of a log file
	Optional	print_each_pop	Print every population to a population file
	Optional	initialization_type	Specify how to initialize the population
	Optional	crossover_type	Select a crossover type for JEGA methods
	Optional	mutation_type	Select a mutation type for JEGA methods
	Optional	seed	Seed of the random number generator
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	model_pointer	Identifier for model block to be used by a method

Description

`moga` stands for Multi-objective Genetic Algorithm, which is a global optimization method that does Pareto optimization for multiple objectives. It supports general constraints and a mixture of real and discrete variables.

Constraints

`moga` can utilize linear constraints using the keywords: `-* linear_inequality_constraint_matrix` `-* linear_inequality_lower_bounds` `-* linear_inequality_upper_bounds` `-* linear_inequality_scale_types` `-* linear_inequality_scales` `* linear_equality_constraint_matrix` `-* linear_equality_targets` `-* linear_equality_scale_types` `-* linear_equality_scales`

Configuration

The genetic algorithm configurations are:

1. fitness
2. replacement
3. niching
4. convergence
5. postprocessor
6. initialization
7. crossover
8. mutation
9. population size

The steps followed by the algorithm are listed below. The configurations will effect how the algorithm completes each step.

Stopping Criteria

The `moga` method respects the `max_iterations` and `max_function_evaluations` method independent controls to provide integer limits for the maximum number of generations and function evaluations, respectively.

The algorithm also stops when convergence is reached. This involves repeated assessment of the algorithm's progress in solving the problem, until some criterion is met.

The specification for convergence in a `moga` can either be `metric_tracker` or can be omitted all together. If omitted, no convergence algorithm will be used and the algorithm will rely on stopping criteria only.

Expected Outputs

The `moga` method respects the `output` method independent control to vary the amount of information presented to the user during execution.

The final results are written to the Dakota tabular output. Additional information is also available - see the `log_file` and `print_each_pop` keywords.

Note that `moga` and `SOGA` create additional output files during execution. "finaldata.dat" is a file that holds the final set of Pareto optimal solutions after any post-processing is complete. "discards.dat" holds solutions that were discarded from the population during the course of evolution.

It can often be useful to plot objective function values from these files to visually see the Pareto front and ensure that finaldata.dat solutions dominate discards.dat solutions. The solutions are written to these output files in the format "Input1...InputN..Output1...OutputM".

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

Important Notes

The pool of potential members is the current population and the current set of offspring.

Choice of fitness assessors is strongly related to the type of replacement algorithm being used and can have a profound effect on the solutions selected for the next generation.

If using the fitness types `layer_rank` or `domination_count`, it is strongly recommended that you use the `replacement_type` `below_limit` (although the roulette wheel selectors can also be used).

The functionality of the `domination_count` selector of JEGA v1.0 can now be achieved using the `domination_count` fitness type and `below_limit` replacement type.

Theory

The basic steps of the `moga` algorithm are as follows:

1. Initialize the population
2. Evaluate the population (calculate the values of the objective function and constraints for each population member)
3. Loop until converged, or stopping criteria reached
 - (a) Perform crossover
 - (b) Perform mutation
 - (c) Evaluate the new population
 - (d) Assess the fitness of each member in the population
 - (e) Replace the population with members selected to continue in the next generation
 - (f) Apply niche pressure to the population
 - (g) Test for convergence
4. Perform post processing

If `moga` is used in a hybrid optimization method (which requires one optimal solution from each individual optimization method to be passed to the subsequent optimization method as its starting point), the solution in the Pareto set closest to the "utopia" point is given as the best solution. This solution is also reported in the Dakota output.

This "best" solution in the Pareto set has minimum distance from the utopia point. The utopia point is defined as the point of extreme (best) values for each objective function. For example, if the Pareto front is bounded by (1,100) and (90,2), then (1,2) is the utopia point. There will be a point in the Pareto set that has minimum L2-norm distance to this point, for example (10,10) may be such a point.

If `moga` is used in a method which may require passing multiple solutions to the next level (such as the `surrogate_based_global` method or `hybrid` methods), the `orthogonal_distance` `postprocessor` type may be used to specify the distances between each solution value to winnow down the solutions in the full Pareto front to a subset which will be passed to the next iteration.

See Also

These keywords may also be of interest:

- [soga](#)

7.2.33.1 `fitness_type`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [fitness_type](#)

Select the fitness type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: domination_count

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Fitness Type (Group 1)	Dakota Keyword layer_rank	Dakota Keyword Description Assign each member to a layer, based on domination the rank based on layers
			domination_count	Rank each member by the number of members that dominate it

Description

The two JEGA methods use different fitness types, which are described on their respective pages.

7.2.33.1.1 [layer_rank](#)

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [fitness_type](#)
- [layer_rank](#)

Assign each member to a layer, based on domination the rank based on layers

Specification

Alias: none

Argument(s): none

Description

The `fitness_type: layer_rank` has been specifically designed to avoid problems with aggregating and scaling objective function values and transforming them into a single objective.

The `layer_rank` fitness assessor works by assigning all non-dominated designs a layer of 0, then from what remains, assigning all the non-dominated a layer of 1, and so on until all designs have been assigned a layer. The values are negated to follow the higher-is-better fitness convention.

Use of the `below_limit` selector with the `layer_rank` fitness assessor has the effect of keeping all those designs whose layer is below a certain threshold again subject to the shrinkage limit.

7.2.33.1.2 `domination_count`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [fitness_type](#)
- [domination_count](#)

Rank each member by the number of members that dominate it

Specification

Alias: none

Argument(s): none

Description

The `fitness_type: domination_count` has been specifically designed to avoid problems with aggregating and scaling objective function values and transforming them into a single objective.

Instead, the `domination_count` fitness assessor works by ordering population members by the negative of the number of designs that dominate them. The values are negated in keeping with the convention that higher fitness is better.

The `layer_rank` fitness assessor works by assigning all non-dominated designs a layer of 0, then from what remains, assigning all the non-dominated a layer of 1, and so on until all designs have been assigned a layer. Again, the values are negated for the higher-is-better fitness convention.

Use of the `below_limit` selector with the `domination_count` fitness assessor has the effect of keeping all designs that are dominated by fewer than a limiting number of other designs subject to the shrinkage limit.

Using it with the `layer_rank` fitness assessor has the effect of keeping all those designs whose layer is below a certain threshold again subject to the shrinkage limit.

7.2.33.2 `replacement_type`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [replacement_type](#)

Select a replacement type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: `below_limit`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Replacement Type (Group 1)	elitist	Use the best designs to form a new population
			roulette_wheel	Replace population
			unique_roulette_wheel	Replace population
			below_limit	Limit number of designs dominating those kept

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs.

In `roulette_wheel` replacement, each design is conceptually allotted a portion of a wheel proportional to its fitness relative to the fitnesses of the other Designs. Then, portions of the wheel are chosen at random and the design occupying those portions are duplicated into the next population. Those Designs allotted larger portions of the wheel are more likely to be selected (potentially many times). `unique_roulette_wheel` replacement is the same as `roulette_wheel` replacement, with the exception that a design may only be selected once. The `below_limit` selector attempts to keep all designs for which the negated fitness is below a certain limit. The values are negated to keep with the convention that higher fitness is better. The inputs to the `below_limit` selector are the limit as a real value, and a `shrinkage_percentage` as a real value. The `shrinkage_percentage` defines the minimum amount of selections that will take place if enough designs are available. It is interpreted as a percentage of the population size that must go on to the subsequent generation. To enforce this, `below_limit` makes all the selections it would make anyway and if that is not enough, it takes the remaining that it needs from the best of what is left (effectively raising its limit as far as it must to get the minimum number of selections). It continues until it has made enough selections. The `shrinkage_percentage` is designed to prevent extreme decreases in the population size at any given generation, and thus prevent a big loss of genetic diversity in a very short time. Without a shrinkage limit, a small group of "super" designs may appear and quickly cull the population down to a size on the order of the limiting value. In this case, all the diversity of the population is lost and it is expensive to re-diversify and spread the population. The

The `replacement_type` for a SOGA may be `roulette_wheel`, `unique_roulette_wheel`, `elitist`, or `favor_feasible`. The `elitist` selector simply chooses the required number of designs taking the most fit. For example, if 100 selections are requested, then the top 100 designs as ranked by fitness will be selected and the remaining will be discarded. The `favor_feasible` replacement type first considers feasibility as a selection criteria. If that does not produce a "winner" then it moves on to considering fitness value. Because of this, any fitness assessor used with the `favor_feasible` selector must only account objectives in the creation of fitness. Therefore, there is such a fitness assessor and its use is enforced when the `favor_feasible` selector is chosen. In that case, and if the output level is set high enough, a message will be presented indicating that the `weighted_sum_only` fitness assessor will be used.

7.2.33.2.1 elitist

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [replacement_type](#)
- [elitist](#)

Use the best designs to form a new population

Specification

Alias: none

Argument(s): none

Description

The `elitist` (default) setting creates a new population using (a) the `replacement_size` best individuals from the current population, (b) and `population_size - replacement_size` individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default `new_solutions_generated` value is set such that the entire set of newly generated individuals will be selected for replacement.

7.2.33.2.2 roulette_wheel

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [replacement_type](#)
- [roulette_wheel](#)

Replace population

Specification

Alias: none

Argument(s): none

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The `replacement_type` of `favor_feasible` is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).

7.2.33.2.3 unique_roulette_wheel

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [replacement_type](#)
- [unique_roulette_wheel](#)

Replace population

Specification

Alias: none

Argument(s): none

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The `replacement_type` of `favor_feasible` is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).

7.2.33.2.4 below_limit

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [replacement_type](#)
- [below_limit](#)

Limit number of designs dominating those kept

Specification

Alias: none

Argument(s): REAL

Default: 6

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		shrinkage_fraction	Decrease the population size by a percentage

Description

The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs.

7.2.33.2.4.1 shrinkage_fraction

- [Keywords Area](#)
- [method](#)

- [moga](#)
- [replacement_type](#)
- [below_limit](#)
- [shrinkage_fraction](#)

Decrease the population size by a percentage

Specification

Alias: shrinkage_percentage

Argument(s): REAL

Default: 0.9

Description

As of JEGA v2.0, all replacement types are common to both MOGA and SOGA. They include the `roulette_wheel`, `unique_roulette_wheel`, `elitist`, and `below_limit` selectors. In `roulette_wheel` replacement, each design is conceptually allotted a portion of a wheel proportional to its fitness relative to the fitnesses of the other Designs. Then, portions of the wheel are chosen at random and the design occupying those portions are duplicated into the next population. Those Designs allotted larger portions of the wheel are more likely to be selected (potentially many times). `unique_roulette_wheel` replacement is the same as `roulette_wheel` replacement, with the exception that a design may only be selected once. The `below_limit` selector attempts to keep all designs for which the negated fitness is below a certain limit. The values are negated to keep with the convention that higher fitness is better. The inputs to the `below_limit` selector are the limit as a real value, and a `shrinkage_percentage` as a real value. The `shrinkage_percentage` defines the minimum amount of selections that will take place if enough designs are available. It is interpreted as a percentage of the population size that must go on to the subsequent generation. To enforce this, `below_limit` makes all the selections it would make anyway and if that is not enough, it takes the remaining that it needs from the best of what is left (effectively raising its limit as far as it must to get the minimum number of selections). It continues until it has made enough selections. The `shrinkage_percentage` is designed to prevent extreme decreases in the population size at any given generation, and thus prevent a big loss of genetic diversity in a very short time. Without a shrinkage limit, a small group of "super" designs may appear and quickly cull the population down to a size on the order of the limiting value. In this case, all the diversity of the population is lost and it is expensive to re-diversify and spread the population. The `elitist` selector simply chooses the required number of designs taking the most fit. For example, if 100 selections are requested, then the top 100 designs as ranked by fitness will be selected and the remaining will be discarded.

7.2.33.3 niching_type

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [niching_type](#)

Specify the type of niching pressure

Specification

Alias: none

Argument(s): none

Default: No niche pressure

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Niching Type (Group 1)	radial	Set niching distance to percentage of non-dominated range
			distance	Enforce minimum Euclidean distance between designs
			max_designs	Limit number of solutions to remain in the population

Description

The purpose of niching is to encourage differentiation along the Pareto frontier and thus a more even and uniform sampling.

This is typically accomplished by discouraging clustering of design points in the performance space. In JEGA, the application of niche pressure occurs as a secondary selection operation. The nicher is given a chance to perform a pre-selection operation prior to the operation of the selection (replacement) operator, and is then called to perform niching on the set of designs that were selected by the selection operator.

The radial nicher takes information input from the user to compute a minimum allowable distance between designs in the performance space and acts as a secondary selection operator whereby it enforces this minimum distance. The distance nicher requires that solutions must be separated from other solutions by a minimum distance in each dimension (vs. Euclidean distance for the radial niching). After niching is complete, all designs in the population will be at least the minimum distance from one another in all directions.

The `radial` niche pressure applicator works by enforcing a minimum Euclidean distance between designs in the performance space at each generation. The algorithm proceeds by starting at the (or one of the) extreme designs along objective dimension 0 and marching through the population removing all designs that are too close to the current design. One exception to the rule is that the algorithm will never remove an extreme design which is defined as a design that is maximal or minimal in all but 1 objective dimension (for a classical 2 objective problem, the extreme designs are those at the tips of the non-dominated frontier). The `distance` nicher enforces a minimum distance in each dimension.

The designs that are removed by the nicher are not discarded. They are buffered and re-inserted into the population during the next pre-selection operation. This way, the selector is still the only operator that discards designs and the algorithm will not waste time "re-filling" gaps created by the nicher.

The `radial` nicher requires as input a vector of fractions with length equal to the number of objectives. The elements of the vector are interpreted as percentages of the non-dominated range for each objective defining a minimum distance to all other designs. All values should be in the range (0, 1). The minimum allowable distance between any two designs in the performance space is the Euclidean (simple square-root-sum-of-squares calculation) distance defined by these percentages. The `distance` nicher has a similar input vector requirement, only the distance is the minimum distance in each dimension.

The `max_designs` niche pressure applicator is designed to choose a limited number of solutions to remain in the population. That number is specified by `num_designs`. It does so in order to balance the tendency for populations to grow very large and thus consuming too many computer resources. It operates by ranking designs according to their fitness standing and a computed count of how many other designs are too close to them. Too

close is a function of the supplied `niche_vector`, which specifies the minimum distance between any two points in the performance space along each dimension individually. Once the designs are all ranked, the top `c\ num_designs` designs are kept in the population and the remaining ones are buffered or discarded. Note that like other niching operators, this one will not discard an extreme design.

7.2.33.3.1 radial

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [niching_type](#)
- [radial](#)

Set niching distance to percentage of non-dominated range

Specification

Alias: none

Argument(s): REALLIST

Default: 0.01 for all objectives

Description

The `radial` nicher requires as input a vector of fractions with length equal to the number of objectives. The elements of the vector are interpreted as percentages of the non-dominated range for each objective defining a minimum distance to all other designs. All values should be in the range (0, 1). The minimum allowable distance between any two designs in the performance space is the Euclidian (simple square-root-sum-of-squares calculation) distance defined by these percentages. The `distance` nicher has a similar input vector requirement, only the distance is the minimum distance in each dimension.

7.2.33.3.2 distance

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [niching_type](#)
- [distance](#)

Enforce minimum Euclidean distance between designs

Specification

Alias: none

Argument(s): REALLIST

Description

Currently, the only niche pressure operators available are the `radial` nicher, the `distance` nicher, and the `max_designs` nicher. The `radial` niche pressure applicator works by enforcing a minimum Euclidean distance between designs in the performance space at each generation. The algorithm proceeds by starting at the (or one of the) extreme designs along objective dimension 0 and marching through the population removing all designs that are too close to the current design. One exception to the rule is that the algorithm will never remove an extreme design which is defined as a design that is maximal or minimal in all but 1 objective dimension (for a classical 2 objective problem, the extreme designs are those at the tips of the non-dominated frontier). The `distance` nicher enforces a minimum distance in each dimension.

The designs that are removed by the nicher are not discarded. They are buffered and re-inserted into the population during the next pre-selection operation. This way, the selector is still the only operator that discards designs and the algorithm will not waste time "re-filling" gaps created by the nicher.

7.2.33.3.3 max_designs

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [niching_type](#)
- [max_designs](#)

Limit number of solutions to remain in the population

Specification

Alias: none

Argument(s): REALLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_designs	Limit the number of solutions

Description

The `max_designs` niche pressure applicator is designed to choose a limited number of solutions to remain in the population. That number is specified by `num_designs`. It does so in order to balance the tendency for populations to grow very large and thus consuming too many computer resources. It operates by ranking designs according to their fitness standing and a computed count of how many other designs are too close to them. Too close is a function of the supplied `niche_vector`, which specifies the minimum distance between any two points in the performance space along each dimension individually. Once the designs are all ranked, the top `c \ num_designs` designs are kept in the population and the remaining ones are buffered or discarded. Note that like other niching operators, this one will not discard an extreme design.

7.2.33.3.3.1 num_designs

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [niching_type](#)

- [max_designs](#)
- [num_designs](#)

Limit the number of solutions

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

The `max_designs` niche pressure applicator is designed to choose a limited number of solutions to remain in the population. That number is specified by `num_designs`. It does so in order to balance the tendency for populations to grow very large and thus consuming too many computer resources. It operates by ranking designs according to their fitness standing and a computed count of how many other designs are too close to them. Too close is a function of the supplied `niche_vector`, which specifies the minimum distance between any two points in the performance space along each dimension individually. Once the designs are all ranked, the top `c \ num_designs` designs are kept in the population and the remaining ones are buffered or discarded. Note that like other niching operators, this one will not discard an extreme design.

7.2.33.4 convergence_type

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [convergence_type](#)

Select the convergence type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: `average_fitness_tracker`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		metric_tracker	Track changes in the non-dominated frontier
	Optional		percent_change	Define the convergence criterion for JEGA methods

	Optional	<code>num_generations</code>	Define the convergence criterion for JEGA methods
--	-----------------	------------------------------	---

Description

The two JEGA methods use different convergence types, which are described on their respective pages.

All the convergence types are modified by the optional keywords `percent_change` and `num_generations`.

7.2.33.4.1 `metric_tracker`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [convergence_type](#)
- [metric_tracker](#)

Track changes in the non-dominated frontier

Specification

Alias: none

Argument(s): none

Default: `metric_tracker`

Description

The moga converger (`metric_tracker`) operates by tracking various changes in the non-dominated frontier from generation to generation. When the changes occurring over a user specified number of generations fall below a user specified threshold, the algorithm stops.

If `metric_tracker` is specified, then a `percent_change` and `num_generations` must be supplied as well. These are listed as optional keywords in the input spec.

Theory

The `metric_tracker` converger tracks 3 metrics specific to the non-dominated frontier from generation to generation. All 3 of these metrics are computed as percent changes between the generations. In order to compute these metrics, the converger stores a duplicate of the non-dominated frontier at each generation for comparison to the non-dominated frontier of the next generation.

The first metric is one that indicates how the expanse of the frontier is changing. The expanse along a given objective is defined by the range of values existing within the non-dominated set. The expansion metric is computed by tracking the extremes of the non-dominated frontier from one generation to the next. Any movement of the extreme values is noticed and the maximum percentage movement is computed as:

$$E_m = \max_{j \text{ of } \text{range}(j, i) - \text{range}(j, i-1)} \frac{\text{abs}(\text{range}(j, i) - \text{range}(j, i-1))}{\text{range}(j, i-1)} \quad j=1, \text{nof}$$

where E_m is the max expansion metric, j is the objective function index, i is the current generation number, and nof is the total number of objectives. The range is the difference between the largest value along an objective and the smallest when considering only non-dominated designs.

The second metric monitors changes in the density of the non-dominated set. The density metric is computed as the number of non-dominated points divided by the hypervolume of the non-dominated region of space. Therefore, changes in the density can be caused by changes in the number of non-dominated points or by changes in size of the non-dominated space or both. The size of the non-dominated space is computed as:

$$Vps(i) = \text{product over } j \text{ of range}(j, i) \quad j=1, \text{nof}$$

where $Vps(i)$ is the hypervolume of the non-dominated space at generation i and all other terms have the same meanings as above.

The density of the a given non-dominated space is then:

$$Dps(i) = Pct(i) / Vps(i)$$

where $Pct(i)$ is the number of points on the non-dominated frontier at generation i .

The percentage increase in density of the frontier is then calculated as

$$Cd = \text{abs}((Dps(i) - Dps(i-1)) / Dps(i-1))$$

where Cd is the change in density metric.

The final metric is one that monitors the "goodness" of the non-dominated frontier. This metric is computed by considering each design in the previous population and determining if it is dominated by any designs in the current population. All that are determined to be dominated are counted. The metric is the ratio of the number that are dominated to the total number that exist in the previous population.

As mentioned above, each of these metrics is a percentage. The tracker records the largest of these three at each generation. Once the recorded percentage is below the supplied percent change for the supplied number of generations consecutively, the algorithm is converged.

7.2.33.4.2 percent_change

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [convergence_type](#)
- [percent_change](#)

Define the convergence criterion for JEGA methods

Specification

Alias: none

Argument(s): REAL

Default: 0.1

Description

The `percent_change` is the threshold beneath which convergence is attained whereby it is compared to the metric value computed.

7.2.33.4.3 num_generations

- [Keywords Area](#)
- [method](#)

- [moga](#)
- [convergence_type](#)
- [num_generations](#)

Define the convergence criterion for JEGA methods

Specification

Alias: none

Argument(s): INTEGER

Default: 10

Description

The `num_generations` is the number of generations over which the metric value should be tracked. Convergence will be attained if the recorded metric is below `percent_change` for `num_generations` consecutive generations.

7.2.33.5 `postprocessor_type`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [postprocessor_type](#)

Post process the final solution from `moga`

Specification

Alias: none

Argument(s): none

Default: No post-processing of solutions

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		orthogonal_distance	Get subset of Pareto front based on distance

Description

The purpose of this operation is to perform any needed data manipulations on the final solution deemed necessary. Currently the `orthogonal_distance` is the only one. It reduces the final solution set size such that a minimum distance in each direction exists between any two designs.

7.2.33.5.1 `orthogonal_distance`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [postprocessor_type](#)
- [orthogonal_distance](#)

Get subset of Pareto front based on distance

Specification

Alias: none

Argument(s): REALLIST

Default: 0.01 for all objectives

Description

Note that MOGA and SOGA create additional output files during execution. "finaldata.dat" is a file that holds the final set of Pareto optimal solutions after any post-processing is complete. "discards.dat" holds solutions that were discarded from the population during the course of evolution. It can often be useful to plot objective function values from these files to visually see the Pareto front and ensure that finaldata.dat solutions dominate discards.dat solutions. The solutions are written to these output files in the format "Input1...InputN..Output1...OutputM". If MOGA is used in a hybrid optimization meta-iteration (which requires one optimal solution from each individual optimization method to be passed to the subsequent optimization method as its starting point), the solution in the Pareto set closest to the "utopia" point is given as the best solution. This solution is also reported in the Dakota output. This "best" solution in the Pareto set has minimum distance from the utopia point. The utopia point is defined as the point of extreme (best) values for each objective function. For example, if the Pareto front is bounded by (1,100) and (90,2), then (1,2) is the utopia point. There will be a point in the Pareto set that has minimum L2-norm distance to this point, for example (10,10) may be such a point. In SOGA, the solution that minimizes the single objective function is returned as the best solution. If moga is used in meta-iteration which may require passing multiple solutions to the next level (such as the `surrogate_based_global` or `hybrid` methods), the `orthogonal_distance` postprocessor type may be used to specify the distances between each solution value to winnow down the solutions in the full Pareto front to a subset which will be passed to the next iteration.

7.2.33.6 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.33.7 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.33.8 scaling

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.33.9 `population_size`

- [Keywords Area](#)
- [method](#)

- [moga](#)
- [population_size](#)

Set the initial population size in JEGA methods

Specification

Alias: none

Argument(s): INTEGER

Default: 50

Description

The number of designs in the initial population is specified by the `population_size`. Note that the `population_size` only sets the size of the initial population. The population size may vary in the JEGA methods according to the type of operators chosen for a particular optimization run.

Also note that some initializers may not strictly respect the initial population size. See the SOGA [flat_file](#) or MOGA [flat_file](#) initializer for an example.

7.2.33.10 `log_file`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [log_file](#)

Specify the name of a log file

Specification

Alias: none

Argument(s): STRING

Default: JEGAGlobal.log

Description

New as of JEGA v2.0 is the introduction of the `log_file` specification. JEGA now uses a logging library to output messages and status to the user. JEGA can be configured at build time to log to both the console window and a text file, one or the other, or neither. The `log_file` input is a string name of a file into which to log. If the build was configured without file logging in JEGA, this input is ignored. If file logging is enabled and no `log_file` is specified, the default file name of JEGAGlobal.log is used.

7.2.33.11 `print_each_pop`

- [Keywords Area](#)
- [method](#)

- [moga](#)
- [print_each_pop](#)

Print every population to a population file

Specification

Alias: none

Argument(s): none

Default: No printing

Description

New to JEGA v2.0 is the introduction of the `print_each_pop` specification. It serves as a flag and if supplied, the population at each generation will be printed to a file named "population<GEN#>.dat" where <GEN#> is the number of the current generation.

7.2.33.12 initialization_type

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [initialization_type](#)

Specify how to initialize the population

Specification

Alias: none

Argument(s): none

Default: unique_random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Initialization Type (Group 1)	simple_random	Create random initial solutions
			unique_random	Create random initial solutions, but enforce uniqueness (default)
			flat_file	Read initial solutions from file

Description

The `initialization_type` defines how the initial population is created for the GA. There are three types:

1. `simple_random`

2. `unique_random` (default)
3. `flat_file`

Setting the size for the `flat_file` initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the `unique_random` initializer and then the `simple_random` initializer if necessary.

7.2.33.12.1 `simple_random`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [initialization_type](#)
- [simple_random](#)

Create random initial solutions

Specification

Alias: none

Argument(s): none

Description

`simple_random` creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs.

7.2.33.12.2 `unique_random`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [initialization_type](#)
- [unique_random](#)

Create random initial solutions, but enforce uniqueness (default)

Specification

Alias: none

Argument(s): none

Description

`unique_random` is the same as `simple_random`, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected.

7.2.33.12.3 flat_file

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [initialization_type](#)
- [flat_file](#)

Read initial solutions from file

Specification

Alias: none

Argument(s): STRING

Description

`flat_file` allows the initial population to be read from a flat file. If `flat_file` is specified, a file name must be given.

Variables can be delimited in the flat file in any way you see fit with a few exceptions. The delimiter must be the same on any given line of input with the exception of leading and trailing whitespace. So a line could look like: 1.1, 2.2 ,3.3 for example but could not look like: 1.1, 2.2 3.3. The delimiter can vary from line to line within the file which can be useful if data from multiple sources is pasted into the same input file. The delimiter can be any string that does not contain any of the characters `.-dDeE` or any of the digits 0-9. The input will be read until the end of the file. The algorithm will discard any configurations for which it was unable to retrieve at least the number of design variables. The objective and constraint entries are not required but if ALL are present, they will be recorded and the design will be tagged as evaluated so that evaluators may choose not to re-evaluate them.

Setting the size for this initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the `unique_random` initializer and then the `simple_random` initializer if necessary. If more designs are found in the initialization files than specified by the initial size, then the initial size is ignored and all designs read out of the files are kept in the initial population.

7.2.33.13 crossover_type

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)

Select a crossover type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: shuffle_random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Crossover Type (Group 1)	multi_point_binary	Use bit switching for crossover events
			multi_point_parameterized_binary	Use bit switching to crossover each design variable
			multi_point_real	Perform crossover in real valued genome
			shuffle_random	Perform crossover by choosing design variable(s)
	Optional		crossover_rate	Specify the probability of a crossover event

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, `N`, of crossover points. This crossover type performs a bit switching crossover at `N` crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at `N` crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at `N` locations. `multi_point_real` crossover performs a variable switching crossover routing at `N` crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

The final crossover type is `shuffle_random`. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

All crossover types take a `crossover_rate`. The crossover rate is used to calculate the number of crossover operations that take place. The number of crossovers is equal to the rate * `population_size`.

7.2.33.13.1 multi_point_binary

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)
- [multi_point_binary](#)

Use bit switching for crossover events

Specification

Alias: none

Argument(s): INTEGER

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

7.2.33.13.2 `multi_point_parameterized_binary`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)
- [multi_point_parameterized_binary](#)

Use bit switching to crossover each design variable

Specification

Alias: none

Argument(s): INTEGER

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

7.2.33.13.3 multi_point_real

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)
- [multi_point_real](#)

Perform crossover in real valued genome

Specification

Alias: none

Argument(s): INTEGER

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

7.2.33.13.4 shuffle_random

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)
- [shuffle_random](#)

Perform crossover by choosing design variable(s)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_parents	Number of parents in random shuffle crossover
	Optional		num_offspring	Number of offspring in random shuffle crossover

Description

The final crossover type is `shuffle_random`. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

7.2.33.13.4.1 num_parents

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)
- [shuffle_random](#)
- [num_parents](#)

Number of parents in random shuffle crossover

Specification

Alias: none

Argument(s): INTEGER

Default: 2

Description

Number of parents in random shuffle crossover

7.2.33.13.4.2 num_offspring

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)
- [shuffle_random](#)
- [num_offspring](#)

Number of offspring in random shuffle crossover

Specification

Alias: none

Argument(s): INTEGER

Default: 2

Description

Number of offspring in random shuffle crossover

7.2.33.13.5 `crossover_rate`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [crossover_type](#)
- [crossover_rate](#)

Specify the probability of a crossover event

Specification

Alias: none

Argument(s): REAL

Default: 0.8

Description

The `crossover_type` controls what approach is employed for combining parent genetic information to create offspring, and the `crossover_rate` specifies the probability of a crossover operation being performed to generate a new offspring. The SCOLIB EA method supports three forms of crossover, `two_point`, `blend`, and `uniform`, which generate a new individual through combinations of two parent individuals. Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate. Uniform crossover creates offspring through random combination of coordinates from the two parents. Blend crossover generates a new individual randomly along the multidimensional vector connecting the two parents.

7.2.33.14 `mutation_type`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)

Select a mutation type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: `replace_uniform`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Mutation Type (Group 1)	bit_random	Mutate by flipping a random bit
			replace_uniform	Use uniformly distributed value over range of parameter
			offset_normal	Set mutation offset to use a normal distribution
			offset_cauchy	Use a Cauchy distribution for the mutation offset
			offset_uniform	Set mutation offset to use a uniform distribution
	Optional		mutation_rate	Set probability of a mutation

Description

Five mutation types are available for selection by keyword: `replace_uniform`, `bit_random`, `offset_cauchy`, `offset_normal`, and `offset_uniform`. They are described in greater detail on their respective keyword pages.

The `offset_*` mutators all act by adding a random "offset" to a variable value. The random amount has a mean of zero in all cases. The size of the offset is controlled using the `mutation_scale` keyword, which is interpreted differently for each `offset_*` type.

The rate of mutations for all types is controlled using the `mutation_rate`. The rate is applied differently in each `mutation_type`.

7.2.33.14.1 `bit_random`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [bit_random](#)

Mutate by flipping a random bit

Specification

Alias: none

Argument(s): none

Description

The `bit_random` mutator introduces random variation by first converting a randomly chosen variable of a randomly chosen design into a binary string. It then flips a randomly chosen bit in the string from a 1 to a 0 or visa versa. In this mutation scheme, the resulting value has more probability of being similar to the original value.

7.2.33.14.2 `replace_uniform`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [replace_uniform](#)

Use uniformly distributed value over range of parameter

Specification

Alias: none

Argument(s): none

Description

`replace_uniform` introduces random variation by first randomly choosing a design variable of a randomly selected design and reassigning it to a random valid value for that variable. No consideration of the current value is given when determining the new value.

7.2.33.14.3 `offset_normal`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [offset_normal](#)

Set mutation offset to use a normal distribution

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	mutation_scale	Scales mutation across range of parameter
--	-----------------	--------------------------------	---

Description

The `offset_normal` mutator introduces random variation by adding a Gaussian random amount to a variable value. The random amount has a standard deviation dependent on the `mutation_scale`.

7.2.33.14.3.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [offset_normal](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.15

Description

The `mutation_scale` is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected `mutation_type`. For `offset_normal` and `offset_cauchy`, `mutation_scale` is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For `offset_uniform`, the range of possible deviation amounts is $\pm 1/2 * (\text{mutation_scale} * \text{variable range})$.

7.2.33.14.4 `offset_cauchy`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [offset_cauchy](#)

Use a Cauchy distribution for the mutation offset

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mutation_scale	Scales mutation across range of parameter

Description

The `offset_cauchy` mutator introduces random variation by adding a Cauchy random amount to a variable value. The random amount has a standard deviation dependent on the `mutation_scale`.

7.2.33.14.4.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [offset_cauchy](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.15

Description

The `mutation_scale` is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected `mutation_type`. For `offset_normal` and `offset_cauchy`, `mutation_scale` is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For `offset_uniform`, the range of possible deviation amounts is $\pm 1/2 * (\text{mutation_scale} * \text{variable range})$.

7.2.33.14.5 `offset_uniform`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [offset_uniform](#)

Set mutation offset to use a uniform distribution

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mutation_scale	Scales mutation across range of parameter

Description

The `offset_uniform` mutator introduces random variation by adding a uniform random amount to a variable value. The random amount depends on the `mutation_scale`.

7.2.33.14.5.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [offset_uniform](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.15

Description

The `mutation_scale` is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected `mutation_type`. For `offset_normal` and `offset_cauchy`, `mutation_scale` is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For `offset_uniform`, the range of possible deviation amounts is $\pm 1/2 * (\text{mutation_scale} * \text{variable range})$.

7.2.33.14.6 `mutation_rate`

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [mutation_type](#)
- [mutation_rate](#)

Set probability of a mutation

Specification

Alias: none

Argument(s): REAL

Default: 0.08

Description

All mutation types have a `mutation_rate`, which controls the number of mutations performed. For `replace_uniform` and all the `offset_*` types, the number of mutations performed is the product of `mutation_rate` and `population_size`. For `bit_random`, it's the product of the `mutation_rate`, number of design variables, and population size

7.2.33.15 seed

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.33.16 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [moga](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.33.17 model_pointer

- [Keywords Area](#)
- [method](#)

- [moga](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'
```

```
method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative
```

```
model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic
```

```
method
  id_method = 'DACE'
```

```

    model_pointer = 'DACE_M'
    sampling sample_type lhs
    samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.34 `soga`

- [Keywords Area](#)
- [method](#)
- [soga](#)

Single-objective Genetic Algorithm (a.k.a Evolutionary Algorithm)

Topics

This keyword is related to the topics:

- [package_jega](#)
- [global_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		fitness_type	Select the fitness type for JEGA methods
	Optional		replacement_type	Select a replacement type for JEGA methods

	Optional	convergence_type	Select the convergence type for JEGA methods
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	population_size	Set the initial population size in JEGA methods
	Optional	log_file	Specify the name of a log file
	Optional	print_each_pop	Print every population to a population file
	Optional	initialization_type	Specify how to initialize the population
	Optional	crossover_type	Select a crossover type for JEGA methods
	Optional	mutation_type	Select a mutation type for JEGA methods
	Optional	seed	Seed of the random number generator
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	model_pointer	Identifier for model block to be used by a method

Description

`soga` stands for Single-objective Genetic Algorithm, which is a global optimization method that supports general constraints and a mixture of real and discrete variables. `soga` is part of the JEGA library.

Constraints `soga` can utilize linear constraints.

Configuration

The genetic algorithm configurations are:

1. fitness
2. replacement
3. convergence
4. initialization
5. crossover
6. mutation
7. population size

The pool of potential members is the current population and the current set of offspring. Choice of fitness assessors is strongly related to the type of replacement algorithm being used and can have a profound effect on the solutions selected for the next generation.

Stopping Criteria

The `soga` method respects the `max_iterations` and `max_function_evaluations` method independent controls to provide integer limits for the maximum number of generations and function evaluations, respectively.

The algorithm also stops when convergence is reached. This involves repeated assessment of the algorithm's progress in solving the problem, until some criterion is met.

Expected Outputs The `soga` method respects the `output` method independent control to vary the amount of information presented to the user during execution.

The final results are written to the Dakota tabular output. Additional information is also available - see the `log_file` and `print_each_pop` keywords.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

Theory

The basic steps of the `soga` algorithm are as follows:

1. Initialize the population
2. Evaluate the population (calculate the values of the objective function and constraints for each population member)
3. Loop until converged, or stopping criteria reached
 - (a) Perform crossover
 - (b) Perform mutation
 - (c) Evaluate the new population
 - (d) Assess the fitness of each member in the population
 - (e) Replace the population with members selected to continue in the next generation
 - (f) Test for convergence

See Also

These keywords may also be of interest:

- [moga](#)

7.2.34.1 fitness_type

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [fitness_type](#)

Select the fitness type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: merit_function

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		merit_function	Balance goals of reducing objective function and satisfying constraints
	Optional		constraint_penalty	Multiplier for the penalty function

Description

The two JEGA methods use different fitness types, which are described on their respective pages.

7.2.34.1.1 merit_function

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [fitness_type](#)
- [merit_function](#)

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none

Argument(s): none

Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

7.2.34.1.2 `constraint_penalty`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [fitness_type](#)
- [constraint_penalty](#)

Multiplier for the penalty function

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

The `merit_function` fitness assessor uses an exterior penalty function formulation to penalize infeasible designs. The specification allows the input of a `constraint_penalty` which is the multiplier to use on the constraint violations.

7.2.34.2 `replacement_type`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [replacement_type](#)

Select a replacement type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: elitist

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required(Choose One)	Replacement Type (Group 1)	elitist	Use the best designs to form a new population
			favor_feasible	Prioritize feasible designs
			roulette_wheel	Replace population
			unique_roulette_wheel	Replace population

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs.

In `roulette_wheel` replacement, each design is conceptually allotted a portion of a wheel proportional to its fitness relative to the fitnesses of the other Designs. Then, portions of the wheel are chosen at random and the design occupying those portions are duplicated into the next population. Those Designs allotted larger portions of the wheel are more likely to be selected (potentially many times). `unique_roulette_wheel` replacement is the same as `roulette_wheel` replacement, with the exception that a design may only be selected once. The `below_limit` selector attempts to keep all designs for which the negated fitness is below a certain limit. The values are negated to keep with the convention that higher fitness is better. The inputs to the `below_limit` selector are the limit as a real value, and a `shrinkage_percentage` as a real value. The `shrinkage_percentage` defines the minimum amount of selections that will take place if enough designs are available. It is interpreted as a percentage of the population size that must go on to the subsequent generation. To enforce this, `below_limit` makes all the selections it would make anyway and if that is not enough, it takes the remaining that it needs from the best of what is left (effectively raising its limit as far as it must to get the minimum number of selections). It continues until it has made enough selections. The `shrinkage_percentage` is designed to prevent extreme decreases in the population size at any given generation, and thus prevent a big loss of genetic diversity in a very short time. Without a shrinkage limit, a small group of "super" designs may appear and quickly cull the population down to a size on the order of the limiting value. In this case, all the diversity of the population is lost and it is expensive to re-diversify and spread the population. The

The `replacement_type` for a SOGA may be `roulette_wheel`, `unique_roulette_wheel`, `elitist`, or `favor_feasible`. The `elitist` selector simply chooses the required number of designs taking the most fit. For example, if 100 selections are requested, then the top 100 designs as ranked by fitness will be selected and the remaining will be discarded. The `favor_feasible` replacement type first considers feasibility as a selection criteria. If that does not produce a "winner" then it moves on to considering fitness value. Because of this, any fitness assessor used with the `favor_feasible` selector must only account objectives in the creation of fitness. Therefore, there is such a fitness assessor and its use is enforced when the `favor_feasible` selector is chosen. In that case, and if the output level is set high enough, a message will be presented indicating that the `weighted_sum_only` fitness assessor will be used.

7.2.34.2.1 elitist

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [replacement_type](#)
- [elitist](#)

Use the best designs to form a new population

Specification

Alias: none

Argument(s): none

Description

The `elitist` (default) setting creates a new population using (a) the `replacement_size` best individuals from the current population, (b) and `population_size - replacement_size` individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default `new_solutions_generated` value is set such that the entire set of newly generated individuals will be selected for replacement.

7.2.34.2.2 `favor_feasible`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [replacement_type](#)
- [favor_feasible](#)

Prioritize feasible designs

Specification

Alias: none

Argument(s): none

Description

This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor.

The `favor_feasible` replacement type first considers feasibility as a selection criteria. If that does not produce a "winner" then it moves on to considering fitness value. Because of this, any fitness assessor used with the `favor_feasible` selector must only account objectives in the creation of fitness. Therefore, there is such a fitness assessor and its use is enforced when the `favor_feasible` selector is chosen. In that case, and if the output level is set high enough, a message will be presented indicating that the `weighted_sum_only` fitness assessor will be used.

7.2.34.2.3 `roulette_wheel`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [replacement_type](#)
- [roulette_wheel](#)

Replace population

Specification

Alias: none

Argument(s): none

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The `replacement_type` of `favor_feasible` is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).

7.2.34.2.4 `unique_roulette_wheel`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [replacement_type](#)
- [unique_roulette_wheel](#)

Replace population

Specification

Alias: none

Argument(s): none

Description

Replace the population with members selected to continue in the next generation. The pool of potential members is the current population and the current set of offspring. The `replacement_type` of `roulette_wheel` or `unique_roulette_wheel` may be used either with MOGA or SOGA problems however they are not recommended for use with MOGA. Given that the only two fitness assessors for MOGA are the `layer_rank` and `domination_count`, the recommended selector is the `below_limit` selector. The `below_limit` replacement will only keep designs that are dominated by fewer than a limiting number of other designs. The `replacement_type` of `favor_feasible` is specific to a SOGA. This replacement operator will always prefer a more feasible design to a less feasible one. Beyond that, it favors solutions based on an assigned fitness value which must have been installed by the weighted sum only fitness assessor (see the discussion below).

7.2.34.3 `convergence_type`

- [Keywords Area](#)
- [method](#)
- [soga](#)

- [convergence_type](#)

Select the convergence type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: average_fitness_tracker

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Convergence Type (Group 1)	best_fitness_tracker	Tracks the best fitness of the population
			average_fitness_tracker	Tracks the average fitness of the population

Description

The two JEGA methods use different convergence types, which are described on their respective pages.

All the convergence types are modified by the optional keywords `percent_change` and `num_generations`.

7.2.34.3.1 [best_fitness_tracker](#)

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [convergence_type](#)
- [best_fitness_tracker](#)

Tracks the best fitness of the population

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		percent_change	Define the convergence criterion for JEGA methods

	Optional	num_generations	Define the convergence criterion for JEGA methods
--	-----------------	---------------------------------	---

Description

The `best_fitness_tracker` tracks the best fitness in the population. Convergence occurs after `num_generations` has passed and there has been less than `percent_change` in the best fitness value. The percent change can be as low as 0% in which case there must be no change at all over the number of generations.

See Also

These keywords may also be of interest:

- [average_fitness_tracker](#)

7.2.34.3.1.1 percent_change

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [convergence_type](#)
- [best_fitness_tracker](#)
- [percent_change](#)

Define the convergence criterion for JEGA methods

Specification

Alias: none

Argument(s): REAL

Default: 0.1

Description

The `percent_change` is the threshold beneath which convergence is attained whereby it is compared to the metric value computed.

7.2.34.3.1.2 num_generations

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [convergence_type](#)
- [best_fitness_tracker](#)
- [num_generations](#)

Define the convergence criterion for JEGA methods

Specification

Alias: none

Argument(s): INTEGER

Default: 10

Description

The `num_generations` is the number of generations over which the metric value should be tracked. Convergence will be attained if the recorded metric is below `percent_change` for `num_generations` consecutive generations.

7.2.34.3.2 `average_fitness_tracker`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [convergence_type](#)
- [average_fitness_tracker](#)

Tracks the average fitness of the population

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		percent_change	Define the convergence criterion for JEGA methods
	Optional		num_generations	Define the convergence criterion for JEGA methods

Description

The `convergence_type` called `average_fitness_tracker` keeps track of the average fitness in a population. If this average fitness does not change more than `percent_change` over some number of generations, `num_generations`, then the solution is reported as converged and the algorithm terminates.

See Also

These keywords may also be of interest:

- [best_fitness_tracker](#)

7.2.34.3.2.1 `percent_change`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [convergence_type](#)
- [average_fitness_tracker](#)
- [percent_change](#)

Define the convergence criterion for JEGA methods

Specification

Alias: none

Argument(s): REAL

Default: 0.1

Description

The `percent_change` is the threshold beneath which convergence is attained whereby it is compared to the metric value computed.

7.2.34.3.2.2 `num_generations`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [convergence_type](#)
- [average_fitness_tracker](#)
- [num_generations](#)

Define the convergence criterion for JEGA methods

Specification

Alias: none

Argument(s): INTEGER

Default: 10

Description

The `num_generations` is the number of generations over which the metric value should be tracked. Convergence will be attained if the recorded metric is below `percent_change` for `num_generations` consecutive generations.

7.2.34.4 max_iterations

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.34.5 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.34.6 scaling

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.34.7 population_size

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [population_size](#)

Set the initial population size in JEGA methods

Specification

Alias: none

Argument(s): INTEGER

Default: 50

Description

The number of designs in the initial population is specified by the `population_size`. Note that the `population_size` only sets the size of the initial population. The population size may vary in the JEGA methods according to the type of operators chosen for a particular optimization run.

Also note that some initializers may not strictly respect the initial population size. See the SOGA [flat_file](#) or MOGA [flat_file](#) initializer for an example.

7.2.34.8 log_file

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [log_file](#)

Specify the name of a log file

Specification

Alias: none

Argument(s): STRING

Default: JEGAGlobal.log

Description

New as of JEGA v2.0 is the introduction of the `log_file` specification. JEGA now uses a logging library to output messages and status to the user. JEGA can be configured at build time to log to both the console window and a text file, one or the other, or neither. The `log_file` input is a string name of a file into which to log. If the build was configured without file logging in JEGA, this input is ignored. If file logging is enabled and no `log_file` is specified, the default file name of JEGAGlobal.log is used.

7.2.34.9 `print_each_pop`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [print_each_pop](#)

Print every population to a population file

Specification

Alias: none

Argument(s): none

Default: No printing

Description

New to JEGA v2.0 is the introduction of the `print_each_pop` specification. It serves as a flag and if supplied, the population at each generation will be printed to a file named "population<GEN#>.dat" where <GEN#> is the number of the current generation.

7.2.34.10 `initialization_type`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [initialization_type](#)

Specify how to initialize the population

Specification

Alias: none

Argument(s): none

Default: `unique_random`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Initialization Type (Group 1)	simple_random	Create random initial solutions
			unique_random	Create random initial solutions, but enforce uniqueness (default)
			flat_file	Read initial solutions from file

Description

The `initialization_type` defines how the initial population is created for the GA. There are three types:

1. `simple_random`
2. `unique_random` (default)
3. `flat_file`

Setting the size for the `flat_file` initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the `unique_random` initializer and then the `simple_random` initializer if necessary.

7.2.34.10.1 `simple_random`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [initialization_type](#)
- [simple_random](#)

Create random initial solutions

Specification

Alias: none

Argument(s): none

Description

`simple_random` creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs.

7.2.34.10.2 `unique_random`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [initialization_type](#)
- [unique_random](#)

Create random initial solutions, but enforce uniqueness (default)

Specification

Alias: none

Argument(s): none

Description

`unique_random` is the same as `simple_random`, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected.

7.2.34.10.3 `flat_file`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [initialization_type](#)
- [flat_file](#)

Read initial solutions from file

Specification

Alias: none

Argument(s): STRING

Description

`flat_file` allows the initial population to be read from a flat file. If `flat_file` is specified, a file name must be given.

Variables can be delimited in the flat file in any way you see fit with a few exceptions. The delimiter must be the same on any given line of input with the exception of leading and trailing whitespace. So a line could look like: 1.1, 2.2 ,3.3 for example but could not look like: 1.1, 2.2 3.3. The delimiter can vary from line to line within the file which can be useful if data from multiple sources is pasted into the same input file. The delimiter can be any string that does not contain any of the characters `.-dDeE` or any of the digits 0-9. The input will be read until the end of the file. The algorithm will discard any configurations for which it was unable to retrieve at least the number of design variables. The objective and constraint entries are not required but if ALL are present, they will be recorded and the design will be tagged as evaluated so that evaluators may choose not to re-evaluate them.

Setting the size for this initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the `unique_random` initializer and then the `simple_random` initializer if necessary. If more designs are found in the initialization files than specified by the initial size, then the initial size is ignored and all designs read out of the files are kept in the initial population.

7.2.34.11 `crossover_type`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)

Select a crossover type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: `shuffle_random`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Crossover Type (Group 1)	multi_point_binary	Use bit switching for crossover events
			multi_point_parameterized_binary	Use bit switching to crossover each design variable
			multi_point_real	Perform crossover in real valued genome
			shuffle_random	Perform crossover by choosing design variable(s)
	Optional		crossover_rate	Specify the probability of a crossover event

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this

assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

The final crossover type is `shuffle_random`. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

All crossover types take a `crossover_rate`. The crossover rate is used to calculate the number of crossover operations that take place. The number of crossovers is equal to the rate * `population_size`.

7.2.34.11.1 `multi_point_binary`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)
- [multi_point_binary](#)

Use bit switching for crossover events

Specification

Alias: none

Argument(s): INTEGER

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

7.2.34.11.2 `multi_point_parameterized_binary`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)
- [multi_point_parameterized_binary](#)

Use bit switching to crossover each design variable

Specification

Alias: none

Argument(s): INTEGER

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

7.2.34.11.3 `multi_point_real`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)
- [multi_point_real](#)

Perform crossover in real valued genome

Specification

Alias: none

Argument(s): INTEGER

Description

There are many crossover types available. `multi_point_binary` crossover requires an integer number, N, of crossover points. This crossover type performs a bit switching crossover at N crossover points in the binary encoded genome of two designs. Thus, crossover may occur at any point along a solution chromosome (in the middle of a gene representing a design variable, for example). `multi_point_parameterized_binary` crossover is similar in that it performs a bit switching crossover routine at N crossover points. However, this crossover type performs crossover on each design variable individually. So the individual chromosomes are crossed at N locations. `multi_point_real` crossover performs a variable switching crossover routing at N crossover points in the real valued genome of two designs. In this scheme, crossover only occurs between design variables (chromosomes). Note that the standard solution chromosome representation in the JEGA algorithm is real encoded and can handle integer or real design variables. For any crossover types that use a binary representation, real variables are converted to long integers by multiplying the real number by 10^6 and then truncating. Note that this assumes a precision of only six decimal places. Discrete variables are represented as integers (indices within a list of possible values) within the algorithm and thus require no special treatment by the binary operators.

7.2.34.11.4 `shuffle_random`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)
- [shuffle_random](#)

Perform crossover by choosing design variable(s)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_parents	Number of parents in random shuffle crossover
	Optional		num_offspring	Number of offspring in random shuffle crossover

Description

The final crossover type is `shuffle_random`. This crossover type performs crossover by choosing design variables at random from a specified number of parents enough times that the requested number of children are produced. For example, consider the case of 3 parents producing 2 children. This operator would go through and for each design variable, select one of the parents as the donor for the child. So it creates a random shuffle of the parent design variable values. The relative numbers of children and parents are controllable to allow for as much mixing as desired. The more parents involved, the less likely that the children will wind up exact duplicates of the parents.

7.2.34.11.4.1 `num_parents`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)
- [shuffle_random](#)
- [num_parents](#)

Number of parents in random shuffle crossover

Specification

Alias: none

Argument(s): INTEGER

Default: 2

Description

Number of parents in random shuffle crossover

7.2.34.11.4.2 num_offspring

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)
- [shuffle_random](#)
- [num_offspring](#)

Number of offspring in random shuffle crossover

Specification

Alias: none

Argument(s): INTEGER

Default: 2

Description

Number of offspring in random shuffle crossover

7.2.34.11.5 crossover_rate

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [crossover_type](#)
- [crossover_rate](#)

Specify the probability of a crossover event

Specification

Alias: none

Argument(s): REAL

Default: 0.8

Description

The `crossover_type` controls what approach is employed for combining parent genetic information to create offspring, and the `crossover_rate` specifies the probability of a crossover operation being performed to generate a new offspring. The SCOLIB EA method supports three forms of crossover, `two_point`, `blend`, and `uniform`, which generate a new individual through combinations of two parent individuals. Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate. Uniform crossover creates offspring through random combination of coordinates from the two parents. Blend crossover generates a new individual randomly along the multidimensional vector connecting the two parents.

7.2.34.12 `mutation_type`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)

Select a mutation type for JEGA methods

Specification

Alias: none

Argument(s): none

Default: `replace_uniform`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Mutation Type (Group 1)	bit_random	Mutate by flipping a random bit
			replace_uniform	Use uniformly distributed value over range of parameter
			offset_normal	Set mutation offset to use a normal distribution
			offset_cauchy	Use a Cauchy distribution for the mutation offset
			offset_uniform	Set mutation offset to use a uniform distribution
	Optional		mutation_rate	Set probability of a mutation

Description

Five mutation types are available for selection by keyword: `replace_uniform`, `bit_random`, `offset_cauchy`, `offset_normal`, and `offset_uniform`. They are described in greater detail on their respective keyword pages.

The `offset_*` mutators all act by adding a random "offset" to a variable value. The random amount has a mean of zero in all cases. The size of the offset is controlled using the `mutation_scale` keyword, which is interpreted differently for each `offset_*` type.

The rate of mutations for all types is controlled using the `mutation_rate`. The rate is applied differently in each `mutation_type`.

7.2.34.12.1 `bit_random`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [bit_random](#)

Mutate by flipping a random bit

Specification

Alias: none

Argument(s): none

Description

The `bit_random` mutator introduces random variation by first converting a randomly chosen variable of a randomly chosen design into a binary string. It then flips a randomly chosen bit in the string from a 1 to a 0 or visa versa. In this mutation scheme, the resulting value has more probability of being similar to the original value.

7.2.34.12.2 `replace_uniform`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [replace_uniform](#)

Use uniformly distributed value over range of parameter

Specification

Alias: none

Argument(s): none

Description

`replace_uniform` introduces random variation by first randomly choosing a design variable of a randomly selected design and reassigning it to a random valid value for that variable. No consideration of the current value is given when determining the new value.

7.2.34.12.3 `offset_normal`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [offset_normal](#)

Set mutation offset to use a normal distribution

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mutation_scale	Scales mutation across range of parameter

Description

The `offset_normal` mutator introduces random variation by adding a Gaussian random amount to a variable value. The random amount has a standard deviation dependent on the `mutation_scale`.

7.2.34.12.3.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [offset_normal](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.15

Description

The `mutation_scale` is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected `mutation_type`. For `offset_normal` and `offset_cauchy`, `mutation_scale` is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For `offset_uniform`, the range of possible deviation amounts is $\pm 1/2 * (\text{mutation_scale} * \text{variable range})$.

7.2.34.12.4 `offset_cauchy`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [offset_cauchy](#)

Use a Cauchy distribution for the mutation offset

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mutation_scale	Scales mutation across range of parameter

Description

The `offset_cauchy` mutator introduces random variation by adding a Cauchy random amount to a variable value. The random amount has a standard deviation dependent on the `mutation_scale`.

7.2.34.12.4.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [offset_cauchy](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.15

Description

The `mutation_scale` is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected `mutation_type`. For `offset_normal` and `offset_cauchy`, `mutation_scale` is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For `offset_uniform`, the range of possible deviation amounts is $\pm 1/2 * (\text{mutation_scale} * \text{variable range})$.

7.2.34.12.5 `offset_uniform`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [offset_uniform](#)

Set mutation offset to use a uniform distribution

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mutation_scale	Scales mutation across range of parameter

Description

The `offset_uniform` mutator introduces random variation by adding a uniform random amount to a variable value. The random amount depends on the `mutation_scale`.

7.2.34.12.5.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [offset_uniform](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.15

Description

The `mutation_scale` is a fraction in the range [0, 1] and is meant to help control the amount of variation that takes place when a variable is mutated. Its behavior depends on the selected `mutation_type`. For `offset_normal` and `offset_cauchy`, `mutation_scale` is multiplied by the range of the variable being mutated to obtain the standard deviation of the offset. For `offset_uniform`, the range of possible deviation amounts is $\pm 1/2 * (\text{mutation_scale} * \text{variable range})$.

7.2.34.12.6 `mutation_rate`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [mutation_type](#)
- [mutation_rate](#)

Set probability of a mutation

Specification

Alias: none

Argument(s): REAL

Default: 0.08

Description

All mutation types have a `mutation_rate`, which controls the number of mutations performed. For `replace_uniform` and all the `offset_*` types, the number of mutations performed is the product of `mutation_rate` and `population_size`. For `bit_random`, it's the product of the `mutation_rate`, number of design variables, and population size

7.2.34.13 `seed`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.34.14 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.34.15 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [soga](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.35 coliny_pattern_search

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)

Pattern search, derivative free optimization method

Topics

This keyword is related to the topics:

- [package_scolib](#)

- [package_coliny](#)
- [global_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		constant_penalty	Use a simple weighted penalty to manage feasibility
	Optional		no_expansion	Don't allow expansion of the search pattern
	Optional		expand_after_success	Set the factor by which a search pattern can be expanded
	Optional		pattern_basis	Pattern basis selection
	Optional		stochastic	Generate trial points in random order
	Optional		total_pattern_size	Total number of points in search pattern
	Optional		exploratory_moves	Exploratory moves selection
	Optional		synchronization	Select how Dakota schedules a batch of concurrent function evaluations in a parallel algorithm
	Optional		contraction_factor	Amount by which step length is rescaled
	Optional		constraint_penalty	Multiplier for the penalty function
	Optional		initial_delta	Initial step size for derivative-free optimizers
	Optional		variable_tolerance	Step length-based stopping criteria for derivative-free optimizers

	Optional	solution_target	Stopping criteria based on objective function value
	Optional	seed	Seed of the random number generator
	Optional	show_misc_options	Show algorithm parameters not exposed in Dakota input
	Optional	misc_options	Set method options not available through Dakota spec
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

Pattern search techniques are nongradient-based optimization methods which use a set of offsets from the current iterate to locate improved points in the design space.

See the page [package_scolib](#) for important information regarding all SCOLIB methods

`coliny_pattern_search` supports concurrency up to the size of the search pattern

Traditional pattern search methods search with a fixed pattern of search directions to try to find improvements to the current iterate. The SCOLIB pattern search methods generalize this simple algorithmic strategy to enable control of how the search pattern is adapted, as well as how each search pattern is evaluated. The `stochastic` and `synchronization` specifications denote how the trial points are evaluated. The `stochastic` specification indicates that the trial points are considered in a random order. For parallel pattern search, `synchronization` dictates whether the evaluations are scheduled using a `blocking` scheduler or a `nonblocking` scheduler. In the `blocking` case, all points in the pattern are evaluated (in parallel), and if the best of these trial points is an improving point, then it becomes the next iterate. These runs are reproducible, assuming use of the same seed in the `stochastic` case. In the `nonblocking` case, all points in the pattern may not be evaluated, since the first improving point found becomes the next iterate. Since the algorithm steps will be subject to parallel timing variability-

ties, these runs will not generally be repeatable. The `synchronization` specification has similar connotations for sequential pattern search. If `blocking` is specified, then each sequential iteration terminates after all trial points have been considered, and if `nonblocking` is specified, then each sequential iteration terminates after the first improving trial point is evaluated. In this release, both `blocking` and `nonblocking` specifications result in blocking behavior (except in the case where `exporatory_moves` below is set to `adaptive_pattern`). Nonblocking behavior will be re-enabled after some underlying technical issues have been resolved.

The particular form of the search pattern is controlled by the `pattern_basis` specification. If `pattern_basis` is `coordinate` basis, then the pattern search uses a plus and minus offset in each coordinate direction, for a total of $2n$ function evaluations in the pattern. This case is depicted in Figure 5.3 for three coordinate dimensions.

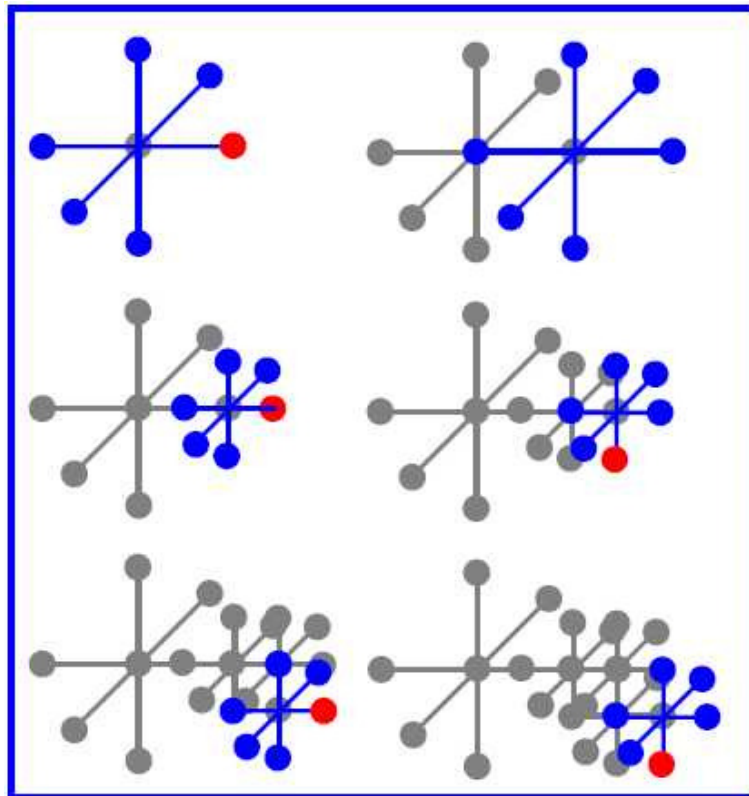


Figure 7.1: Depiction of coordinate pattern search algorithm

If `pattern_basis` is `simplex`, then pattern search uses a minimal positive basis simplex for the parameter space, for a total of $n+1$ function evaluations in the pattern. Note that the `simplex` pattern basis can be used for unbounded problems only. The `total_pattern_size` specification can be used to augment the basic `coordinate` and `simplex` patterns with additional function evaluations, and is particularly useful for parallel load balancing. For example, if some function evaluations in the pattern are dropped due to duplication or bound constraint interaction, then the `total_pattern_size` specification instructs the algorithm to generate new offsets to bring the total number of evaluations up to this consistent total.

The `exploratory_moves` specification controls how the search pattern is adapted. (The search pattern can be adapted after an improving trial point is found, or after all trial points in a search pattern have been found to be unimproving points.) The following exploratory moves selections are supported by SCOLIB:

- The `basic_pattern` case is the simple pattern search approach, which uses the same pattern in each iteration.
- The `multi_step` case examines each trial step in the pattern in turn. If a successful step is found, the pattern search continues examining trial steps about this new point. In this manner, the effects of multiple successful steps are cumulative within a single iteration. This option does not support any parallelism and will result in a serial pattern search.

- The `adaptive_pattern` case invokes a pattern search technique that adaptively rescales the different search directions to maximize the number of redundant function evaluations. See [46] for details of this method. In preliminary experiments, this method had more robust performance than the standard `basic_pattern` case in serial tests. This option supports a limited degree of parallelism. After successful iterations (where the step length is not contracted), a parallel search will be performed. After unsuccessful iterations (where the step length is contracted), only a single evaluation is performed.

The `initial_delta` and `variable_tolerance` specifications provide the initial offset size and the threshold size at which to terminate the algorithm. For any dimension that has both upper and lower bounds, this step length will be internally rescaled to provide search steps of length `initial_delta * range * 0.1`. This rescaling does not occur for other dimensions, so search steps in those directions have length `initial_delta`. Note that the factor of 0.1 in the rescaling could result in an undesirably small initial step. This can be offset by providing a large `initial_delta`.

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value `contraction_factor`, and they expand the step length by the value `(1/contraction_factor)`. The `expand_after_success` control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the `no_expansion` flag instructs the algorithm to forgo pattern expansion altogether.

Finally, constraint infeasibility can be managed in a somewhat more sophisticated manner than the simple weighted penalty function. If the `constant_penalty` specification is used, then the simple weighted penalty scheme described above is used. Otherwise, the constraint penalty is adapted to the value `constraint_penalty/L`, where L is the the smallest step length used so far.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

See Also

These keywords may also be of interest:

- [coliny_beta](#)
- [coliny_direct](#)
- [coliny_cobyla](#)
- [coliny_ea](#)
- [coliny_solis_wets](#)

7.2.35.1 `constant_penalty`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [constant_penalty](#)

Use a simple weighted penalty to manage feasibility

Specification

Alias: none

Argument(s): none

Default: algorithm dynamically adapts the constraint penalty

Description

Finally, constraint infeasibility can be managed in a somewhat more sophisticated manner than the simple weighted penalty function. If the `constant_penalty` specification is used, then the simple weighted penalty scheme described above is used. Otherwise, the constraint penalty is adapted to the value `constraint_penalty/L`, where L is the the smallest step length used so far.

7.2.35.2 no_expansion

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [no_expansion](#)

Don't allow expansion of the search pattern

Specification

Alias: none

Argument(s): none

Default: algorithm may expand pattern size

Description

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value `contraction_factor`, and they expand the step length by the value $(1/\text{contraction_factor})$. The `expand_after_success` control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the `no_expansion` flag instructs the algorithm to forgo pattern expansion altogether.

7.2.35.3 expand_after_success

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [expand_after_success](#)

Set the factor by which a search pattern can be expanded

Specification

Alias: none

Argument(s): INTEGER

Default: 5

Description

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value `contraction_factor`, and they expand the step length by the value $(1/\text{contraction_factor})$. The `expand_after_success` control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the `no_expansion` flag instructs the algorithm to forgo pattern expansion altogether.

7.2.35.4 `pattern_basis`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [pattern_basis](#)

Pattern basis selection

Specification

Alias: none

Argument(s): none

Default: coordinate

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Pattern Basis Type (Group 1)	coordinate	Use coordinate directions as search pattern
			simplex	Use a minimal simplex for the search pattern

Description

The particular form of the search pattern is controlled by the `pattern_basis` specification. If `pattern_basis` is `coordinate` basis, then the pattern search uses a plus and minus offset in each coordinate direction, for a total of $2n$ function evaluations in the pattern. This case is depicted in Figure 5.3 for three coordinate dimensions.

7.2.35.4.1 `coordinate`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)

- [pattern_basis](#)
- [coordinate](#)

Use coordinate directions as search pattern

Specification

Alias: none

Argument(s): none

Description

The particular form of the search pattern is controlled by the `pattern_basis` specification. If `pattern_basis` is `coordinate` basis, then the pattern search uses a plus and minus offset in each coordinate direction, for a total of $2n$ function evaluations in the pattern. This case is depicted in Figure 5.3 for three coordinate dimensions.

7.2.35.4.2 simplex

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [pattern_basis](#)
- [simplex](#)

Use a minimal simplex for the search pattern

Specification

Alias: none

Argument(s): none

Description

If `pattern_basis` is `simplex`, then pattern search uses a minimal positive basis simplex for the parameter space, for a total of $n+1$ function evaluations in the pattern. Note that the `simplex` pattern basis can be used for unbounded problems only. The `total_pattern_size` specification can be used to augment the basic `coordinate` and `simplex` patterns with additional function evaluations, and is particularly useful for parallel load balancing. For example, if some function evaluations in the pattern are dropped due to duplication or bound constraint interaction, then the `total_pattern_size` specification instructs the algorithm to generate new offsets to bring the total number of evaluations up to this consistent total.

7.2.35.5 stochastic

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [stochastic](#)

Generate trial points in random order

Specification

Alias: none

Argument(s): none

Description

Traditional pattern search methods search with a fixed pattern of search directions to try to find improvements to the current iterate. The SCOLIB pattern search methods generalize this simple algorithmic strategy to enable control of how the search pattern is adapted, as well as how each search pattern is evaluated. The `stochastic` and `synchronization` specifications denote how the trial points are evaluated. The `stochastic` specification indicates that the trial points are considered in a random order. For parallel pattern search, `synchronization` dictates whether the evaluations are scheduled using a `blocking` scheduler or a `nonblocking` scheduler (i.e.,

7.2.35.6 `total_pattern_size`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [total_pattern_size](#)

Total number of points in search pattern

Specification

Alias: none

Argument(s): INTEGER

Default: no augmentation of basic pattern

Description

If `pattern_basis` is `simplex`, then pattern search uses a minimal positive basis simplex for the parameter space, for a total of $n+1$ function evaluations in the pattern. Note that the `simplex` pattern basis can be used for unbounded problems only. The `total_pattern_size` specification can be used to augment the basic `coordinate` and `simplex` patterns with additional function evaluations, and is particularly useful for parallel load balancing. For example, if some function evaluations in the pattern are dropped due to duplication or bound constraint interaction, then the `total_pattern_size` specification instructs the algorithm to generate new offsets to bring the total number of evaluations up to this consistent total.

7.2.35.7 `exploratory_moves`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [exploratory_moves](#)

Exploratory moves selection

Specification

Alias: none

Argument(s): none

Default: basic_pattern

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Exploration Mode (Group 1)	multi_step	Examine trial step around successful new point
			adaptive_pattern	Adaptively rescale search directions
			basic_pattern	Use the same search pattern every iteration

Description

The `exploratory_moves` specification controls how the search pattern is adapted. (The search pattern can be adapted after an improving trial point is found, or after all trial points in a search pattern have been found to be unimproving points.) The following exploratory moves selections are supported by SCOLIB:

7.2.35.7.1 multi_step

- [Keywords Area](#)
- [method](#)
- [colony_pattern_search](#)
- [exploratory_moves](#)
- [multi_step](#)

Examine trial step around successful new point

Specification

Alias: none

Argument(s): none

Description

The `multi_step` case examines each trial step in the pattern in turn. If a successful step is found, the pattern search continues examining trial steps about this new point. In this manner, the effects of multiple successful steps are cumulative within a single iteration. This option does not support any parallelism and will result in a serial pattern

7.2.35.7.2 adaptive_pattern

- [Keywords Area](#)
- [method](#)
- [colony_pattern_search](#)

- [exploratory_moves](#)
- [adaptive_pattern](#)

Adaptively rescale search directions

Specification

Alias: none

Argument(s): none

Description

The `adaptive_pattern` case invokes a pattern search technique that adaptively rescales the different search directions to maximize the number of redundant function evaluations. See [46] for details of this method. In preliminary experiments, this method had more robust performance than the standard `basic_pattern` case in serial tests. This option supports a limited degree of parallelism. After successful iterations (where the step length is not contracted), a parallel search will be performed. After unsuccessful iterations (where the step length is contracted), only a single evaluation is performed.

7.2.35.7.3 `basic_pattern`

- [Keywords Area](#)
- [method](#)
- [colony_pattern_search](#)
- [exploratory_moves](#)
- [basic_pattern](#)

Use the same search pattern every iteration

Specification

Alias: none

Argument(s): none

Description

The `basic_pattern` case is the simple pattern search approach, which uses the same pattern in each iteration.

7.2.35.8 `synchronization`

- [Keywords Area](#)
- [method](#)
- [colony_pattern_search](#)
- [synchronization](#)

Select how Dakota schedules a batch of concurrent function evaluations in a parallel algorithm

Specification

Alias: none

Argument(s): none

Default: method specific

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Synchronization (Group 1)	Dakota Keyword blocking	Dakota Keyword Description Complete concurrent evaluation of all evaluations within a batch
			nonblocking	A batch of concurrent evaluations are queried for completions and a partial set is returned to the algorithm

Description

The `synchronization` specification can be used to specify the use of either `blocking` or `nonblocking` schedulers, where the former must block until all evaluations are completed and the latter does not block, returning only the partial set that has been completed for a particular call to the scheduler.

7.2.35.8.1 `blocking`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [synchronization](#)
- [blocking](#)

Complete concurrent evaluation of all evaluations within a batch

Specification

Alias: none

Argument(s): none

Description

In the `blocking` case, all of the concurrent evaluations in the batch must be completed and all results are returned together to the algorithm.

These runs are reproducible, assuming use of the same seed in cases with random number generation.

7.2.35.8.2 nonblocking

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [synchronization](#)
- [nonblocking](#)

A batch of concurrent evaluations are queried for completions and a partial set is returned to the algorithm

Specification

Alias: none

Argument(s): none

Description

In the `nonblocking` case, the set of running evaluations is queried for completion and a partial set of results is returned to an asynchronous algorithm, which can then generate new evaluations based on these partial results.

In all cases, the algorithm steps will be subject to parallel timing variabilities / race conditions such that these runs will not generally be repeatable regardless of seed specification.

7.2.35.9 contraction_factor

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [contraction_factor](#)

Amount by which step length is rescaled

Specification

Alias: none

Argument(s): REAL

Default: 0.5

Description

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction_factor}$

7.2.35.10 `constraint_penalty`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [constraint_penalty](#)

Multiplier for the penalty function

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

7.2.35.11 `initial_delta`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [initial_delta](#)

Initial step size for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 1.0 (COBYLA), 0.1*range (PS, SW)

Description

The `initial_delta` keyword defines the size of the first search step in derivative-free optimization methods, specifically `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`. It is applied in an absolute sense to all search directions.

Default Behavior

The default value is 1.0.

Usage Tips

It is recommended that `initial_delta` be the approximate distance from the initial point to the solution. If this distance is unknown, it is advisable to err on the side of choosing an `initial_delta` that is too large or to not specify it. Relative application of `initial_delta` is not available unless the user scales the problem accordingly.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    initial_delta = .5
    contraction_factor = 0.25
    merit_function merit1_smooth
    smoothing_factor = 1.0
    constraint_tolerance = 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    initial_delta = 2.0
    seed = 1234
```

7.2.35.12 `variable_tolerance`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [variable_tolerance](#)

Step length-based stopping criteria for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-4 (COBYLA), 1.0e-5 (PS), 1.0e-6 (SW)

Description

The `variable_tolerance` keyword defines the minimum step length allowed by the optimizer and is used to determine convergence. It is applicable to `asynch_pattern_search`, `coliny_cobyla`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`.

Default Behavior

The default value varies according to method as follows:

- `asynch_pattern_search`: 1.0e-2
- `coliny_cobyla`: 1.0e-4
- `coliny_pattern_search`: 1.0e-5
- `coliny_solis_wets`: 1.0e-6
- `mesh_adaptive_search`: 1.0e-6

Usage Tips

It is recommended that `variable_tolerance` be set to a value for which changes of that scale in parameter values cause negligible changes in the objective function.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    contraction_factor = 0.25
    variable_tolerance = 1.e-4
    solution_target = 1.e-6
    max_function_evaluations 500
    constraint_tolerance 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    variable_tolerance = 0.01
    seed = 1234
```

7.2.35.13 `solution_target`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: solution_accuracy

Argument(s): REAL

Default: -DBL_MAX

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.35.14 seed

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```


7.2.35.15 show_misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [show_misc_options](#)

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

Argument(s): none

Default: no dump of specification options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.35.16 misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [misc_options](#)

Set method options not available through Dakota spec

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB

methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.35.17 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.35.18 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)

- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.35.19 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.35.20 `scaling`

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.35.21 model_pointer

- [Keywords Area](#)
- [method](#)
- [coliny_pattern_search](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.36 coliny_solis_wets

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)

Simple greedy local search method

Topics

This keyword is related to the topics:

- [package_scolib](#)
- [package_coliny](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		contract_after_failure	The number of unsuccessful cycles prior to contraction.
	Optional		no_expansion	Don't allow expansion of the search pattern
	Optional		expand_after_success	Set the factor by which a search pattern can be expanded
	Optional		constant_penalty	Use a simple weighted penalty to manage feasibility
	Optional		contraction_factor	Amount by which step length is rescaled
	Optional		constraint_penalty	Multiplier for the penalty function
	Optional		initial_delta	Initial step size for derivative-free optimizers
	Optional		variable_tolerance	Step length-based stopping criteria for derivative-free optimizers
	Optional		solution_target	Stopping criteria based on objective function value
	Optional		seed	Seed of the random number generator
	Optional		show_misc_options	Show algorithm parameters not exposed in Dakota input
	Optional		misc_options	Set method options not available through Dakota spec

	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The Solis-Wets method is a simple greedy local search heuristic for continuous parameter spaces. Solis-Wets generates trial points using a multivariate normal distribution, and unsuccessful trial points are reflected about the current point to find a descent direction.

See the page [package_scolib](#) for important information regarding all SCOLIB methods

`coliny_solis_wets` is inherently serial, no concurrency is used.

These specifications have the same meaning as corresponding specifications for [coliny_pattern_search](#). Please see that page for specification details.

In particular, `coliny_solis_wets` supports dynamic rescaling of the step length, and dynamic rescaling of the constraint penalty. The only new specification is `contract_after_failure`, which specifies the number of unsuccessful cycles which must occur with a specific delta prior to contraction of the delta.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

See Also

These keywords may also be of interest:

- [coliny_beta](#)

- [coliny_direct](#)
- [coliny_pattern_search](#)
- [coliny_cobyla](#)
- [coliny_ea](#)

7.2.36.1 [contract_after_failure](#)

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [contract_after_failure](#)

The number of unsuccessful cycles prior to contraction.

Specification

Alias: none

Argument(s): INTEGER

Default: 4*number of variables

Description

In particular, [coliny_solis_wets](#) supports dynamic rescaling of the step length, and dynamic rescaling of the constraint penalty. The only new specification is [contract_after_failure](#), which specifies the number of unsuccessful cycles which must occur with a specific delta prior to contraction of the delta.

7.2.36.2 [no_expansion](#)

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [no_expansion](#)

Don't allow expansion of the search pattern

Specification

Alias: none

Argument(s): none

Default: algorithm may expand pattern size

Description

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value `contraction_factor`, and they expand the step length by the value $(1/\text{contraction_factor})$. The `expand_after_success` control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the `no_expansion` flag instructs the algorithm to forgo pattern expansion altogether.

7.2.36.3 `expand_after_success`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [expand_after_success](#)

Set the factor by which a search pattern can be expanded

Specification

Alias: none

Argument(s): INTEGER

Default: 5

Description

In general, pattern search methods can expand and contract their step lengths. SCOLIB pattern search methods contract the step length by the value `contraction_factor`, and they expand the step length by the value $(1/\text{contraction_factor})$. The `expand_after_success` control specifies how many successful objective function improvements must occur with a specific step length prior to expansion of the step length, whereas the `no_expansion` flag instructs the algorithm to forgo pattern expansion altogether.

7.2.36.4 `constant_penalty`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [constant_penalty](#)

Use a simple weighted penalty to manage feasibility

Specification

Alias: none

Argument(s): none

Default: algorithm dynamically adapts the constraint penalty

Description

Finally, constraint infeasibility can be managed in a somewhat more sophisticated manner than the simple weighted penalty function. If the `constant_penalty` specification is used, then the simple weighted penalty scheme described above is used. Otherwise, the constraint penalty is adapted to the value `constraint_penalty/L`, where L is the the smallest step length used so far.

7.2.36.5 `contraction_factor`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [contraction_factor](#)

Amount by which step length is rescaled

Specification

Alias: none

Argument(s): REAL

Default: 0.5

Description

For pattern search methods, `contraction_factor` specifies the amount by which step length is rescaled after unsuccessful iterates, must be strictly between 0 and 1.

For methods that can expand the step length, the expansion is $1/\text{contraction_factor}$

7.2.36.6 `constraint_penalty`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [constraint_penalty](#)

Multiplier for the penalty function

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

7.2.36.7 `initial_delta`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [initial_delta](#)

Initial step size for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 1.0 (COBYLA), 0.1*range (PS, SW)

Description

The `initial_delta` keyword defines the size of the first search step in derivative-free optimization methods, specifically `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`. It is applied in an absolute sense to all search directions.

Default Behavior

The default value is 1.0.

Usage Tips

It is recommended that `initial_delta` be the approximate distance from the initial point to the solution. If this distance is unknown, it is advisable to err on the side of choosing an `initial_delta` that is too large or to not specify it. Relative application of `initial_delta` is not available unless the user scales the problem accordingly.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    initial_delta = .5
    contraction_factor = 0.25
    merit_function merit1_smooth
    smoothing_factor = 1.0
    constraint_tolerance = 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
```

```

initial_delta = .2
variable_tolerance = 1.e-4
max_iterations 100
solution_accuracy = 1.e-6
seed = 1234
max_function_evaluations = 1000

```

For `mesh_adaptive_search`

```

method
  mesh_adaptive_search
    initial_delta = 2.0
    seed = 1234

```

7.2.36.8 variable_tolerance

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [variable_tolerance](#)

Step length-based stopping criteria for derivative-free optimizers

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-4 (COBYLA), 1.0e-5 (PS), 1.0e-6 (SW)

Description

The `variable_tolerance` keyword defines the minimum step length allowed by the optimizer and is used to determine convergence. It is applicable to `asynch_pattern_search`, `coliny_cobyala`, `coliny_pattern_search`, `coliny_solis_wets`, and `mesh_adaptive_search`.

Default Behavior

The default value varies according to method as follows:

- `asynch_pattern_search`: 1.0e-2
- `coliny_cobyala`: 1.0e-4
- `coliny_pattern_search`: 1.0e-5
- `coliny_solis_wets`: 1.0e-6
- `mesh_adaptive_search`: 1.0e-6

Usage Tips

It is recommended that `variable_tolerance` be set to a value for which changes of that scale in parameter values cause negligible changes in the objective function.

Examples

Three example method input blocks appear below.

For `asynch_pattern_search`:

```
method
  asynch_pattern_search
    contraction_factor = 0.25
    variable_tolerance = 1.e-4
    solution_target = 1.e-6
    max_function_evaluations 500
    constraint_tolerance 1.e-6
```

For `coliny_pattern_search`:

```
method
  coliny_pattern_search
    initial_delta = .2
    variable_tolerance = 1.e-4
    max_iterations 100
    solution_accuracy = 1.e-6
    seed = 1234
    max_function_evaluations = 1000
```

For `mesh_adaptive_search`

```
method
  mesh_adaptive_search
    variable_tolerance = 0.01
    seed = 1234
```

7.2.36.9 `solution_target`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: `-DBL_MAX`

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.36.10 `seed`

- [Keywords Area](#)
- [method](#)

- [coliny_solis_wets](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.36.11 `show_misc_options`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [show_misc_options](#)

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

Argument(s): none

Default: no dump of specification options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.36.12 `misc_options`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [misc_options](#)

Set method options not available through Dakota spec

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.36.13 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.36.14 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.36.15 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.36.16 scaling

- [Keywords Area](#)
- [method](#)
- [coliny_solis_wets](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.36.17 `model_pointer`

- [Keywords Area](#)
- [method](#)

- [coliny_solis_wets](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
```

```
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.37 coliny_cobyla

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)

Constrained Optimization BY Linear Approximations (COBYLA)

Topics

This keyword is related to the topics:

- [package_scolib](#)
- [package_coliny](#)
- [local_optimization_methods](#)
- [constrained](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_delta	Reasonable initial changes to optimization variables
	Optional		variable_tolerance	Required or expected accuracy in optimization variables.
	Optional		solution_target	Stopping criteria based on objective function value
	Optional		seed	Seed of the random number generator
	Optional		show_misc_options	Show algorithm parameters not exposed in Dakota input
	Optional		misc_options	Set method options not available through Dakota spec
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

The Constrained Optimization BY Linear Approximations (COBYLA) algorithm is an extension to the Nelder-Mead simplex algorithm for handling general linear/nonlinear constraints and is invoked using the `coliny_cobyala` group specification. The COBYLA algorithm employs linear approximations to the objective and constraint functions, the approximations being formed by linear interpolation at $N+1$ points in the space of the variables. We regard these interpolation points as vertices of a simplex. The step length parameter controls the size of the simplex and it is reduced automatically from `initial_delta` to `variable_tolerance`. One advantage that COBYLA has over many of its competitors is that it treats each constraint individually when calculating a change to the variables, instead of lumping the constraints together into a single penalty function.

See the page [package_scolib](#) for important information regarding all SCOLIB methods

`coliny_cobyala` is inherently serial.

Stopping Criteria

COBYLA currently only supports termination based on

- [max_function_evaluations](#)
- [solution_target](#)

Other method-independent stopping criteria (`max_iterations` and `convergence_tolerance`) will be ignored if set.

Known Bugs

The implementation of the `coliny_cobyala` optimization method is such that the best function value is not always returned to Dakota for reporting. The user is advised to look through the Dakota screen output or the tabular output file (if generated) to confirm what the best function value and corresponding parameter values are.

The `coliny_cobyala` optimization method does not always respect bound constraints when scaling is turned on.

Neither bug will be fixed, as maintaining third-party source code (such as COBYLA) is outside of the Dakota project scope.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

See Also

These keywords may also be of interest:

- [coliny_beta](#)
- [coliny_direct](#)
- [coliny_pattern_search](#)

- [coliny_ea](#)
- [coliny_solis_wets](#)

7.2.37.1 `initial_delta`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [initial_delta](#)

Reasonable initial changes to optimization variables

Specification

Alias: none

Argument(s): REAL

Default: 1.0 (COBYLA), 0.1*range (PS, SW)

Description

`initial_delta` for COBYLA should be set to be a value that is reasonable for initial changes to the optimization variables. It represents a distance from the initial simplex within which linear approximation subproblems can be trusted to be sufficiently representative of the true problem. It is analogous to the initial trust-region size used in trust-region methods.

Default Behavior

The default value is 1.0.

7.2.37.2 `variable_tolerance`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [variable_tolerance](#)

Required or expected accuracy in optimization variables.

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-4 (COBYLA), 1.0e-5 (PS), 1.0e-6 (SW)

Description

Unlike its use as a stopping criteria in other methods, `variable_tolerance` is used to communicate the accuracy of the optimization variables to COBYLA. It represents the minimum distance from the simplex within which linear approximation subproblems can be trusted to be sufficiently representative of the true problem. It is analogous to the minimum trust-region size used in trust-region methods. Note that, per COBYLA documentation, the level of accuracy is not guaranteed.

Default Behavior

The default value is 0.0001.

Additional Discussion

While `variable_tolerance` is not a stopping criteria for COBYLA, we note that it may have a side effect regarding convergence. In particular, if lower accuracy is required of the optimization variables, it may stop sooner than if higher accuracy is required.

7.2.37.3 `solution_target`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyala](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: `-DBL_MAX`

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.37.4 `seed`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyala](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.37.5 show_misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [show_misc_options](#)

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

Argument(s): none

Default: no dump of specification options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The

`misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.37.6 `misc_options`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [misc_options](#)

Set method options not available through Dakota spec

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.37.7 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.37.8 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.37.9 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.37.10 scaling

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto`, `log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.37.11 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [coliny_cobyla](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.38 coliny_direct

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)

Dividing RECTangles method

Topics

This keyword is related to the topics:

- [package_scolib](#)
- [package_coliny](#)
- [global_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		division	Determine how rectangles are subdivided
	Optional		global_balance_-parameter	Tolerance for whether a subregion is worth dividing
	Optional		local_balance_-parameter	Tolerance for whether a subregion is worth dividing

	Optional	max_boxsize_limit	Stopping Criterion based on longest edge of hyperrectangle
	Optional	min_boxsize_limit	Stopping Criterion based on shortest edge of hyperrectangle
	Optional	constraint_penalty	Multiplier for the penalty function
	Optional	solution_target	Stopping criteria based on objective function value
	Optional	seed	Seed of the random number generator
	Optional	show_misc_options	Show algorithm parameters not exposed in Dakota input
	Optional	misc_options	Set method options not available through Dakota spec
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The Dividing RECTangles (DIRECT) optimization algorithm is a derivative free global optimization method that balances local search in promising regions of the design space with global search in unexplored regions. As shown in Figure 5.1, DIRECT adaptively subdivides the space of feasible design points so as to guarantee that iterates are generated in the neighborhood of a global minimum in finitely many iterations.

DIRECT" \image latex direct1.eps "Design space partitioning with DIRECT" width=10cm

In practice, DIRECT has proven an effective heuristic for engineering design applications, for which it is able to quickly identify candidate solutions that can be further refined with fast local optimizers.

See the page [package_scolib](#) for important information regarding all SCOLIB methods

The DIRECT algorithm supports concurrency up to twice the number of variables being optimized.

DIRECT uses the `solution_target`, `constraint_penalty` and `show_misc_options` specifications that are described in [package_scolib](#). Note, however, that DIRECT uses a fixed penalty value for constraint violations (i.e. it is not dynamically adapted as is done in `coliny_pattern_search`).

Search Parameters

The `global_balance_parameter` controls how much global search is performed by only allowing a subregion to be subdivided if the size of the subregion divided by the size of the largest subregion is at least `global_balance_parameter`. Intuitively, this forces large subregions to be subdivided before the smallest subregions are refined. The `local_balance_parameter` provides a tolerance for estimating whether the smallest subregion can provide a sufficient decrease to be worth subdividing; the default value is a small value that is suitable for most applications.

Stopping Criteria

DIRECT can be terminated with:

- [max_function_evaluations](#)
- [max_iterations](#)
- [convergence_tolerance](#)
- [solution_target](#)
- [max_boxsize_limit](#)
- [min_boxsize_limit](#) - most effective in practice

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when [calibration_terms](#) are specified)

See Also

These keywords may also be of interest:

- [coliny_beta](#)
- [coliny_pattern_search](#)
- [coliny_cobyla](#)
- [coliny_ea](#)
- [coliny_solis_wets](#)
- [ncsu_direct](#)

7.2.38.1 division

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [division](#)

Determine how rectangles are subdivided

Specification

Alias: none

Argument(s): none

Default: major_dimension

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Divide Along (Group 1)	Dakota Keyword major_dimension	Dakota Keyword Description (default) Longest edge of subregion is subdivided
			all_dimensions	All dimensions are simultaneously subdivided

Description

The `division` specification determines how DIRECT subdivides each subregion of the search space.

If `division` is set to `major_dimension`, then the dimension representing the longest edge of the subregion is subdivided (this is the default). If `division` is set to `all_dimensions`, then all dimensions are simultaneously subdivided.

7.2.38.1.1 major_dimension

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [division](#)
- [major_dimension](#)

(default) Longest edge of subregion is subdivided

Specification

Alias: none

Argument(s): none

Description

Longest edge of subregion is subdivided

7.2.38.1.2 all_dimensions

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [division](#)
- [all_dimensions](#)

All dimensions are simultaneously subdivided

Specification

Alias: none

Argument(s): none

Description

All dimensions are simultaneously subdivided

7.2.38.2 global_balance_parameter

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [global_balance_parameter](#)

Tolerance for whether a subregion is worth dividing

Specification

Alias: none

Argument(s): REAL

Default: 0.0

Description

The `global_balance_parameter` controls how much global search is performed by only allowing a subregion to be subdivided if the size of the subregion divided by the size of the largest subregion is at least `global_balance_parameter`. Intuitively, this forces large subregions to be subdivided before the smallest subregions are refined.

7.2.38.3 `local_balance_parameter`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [local_balance_parameter](#)

Tolerance for whether a subregion is worth dividing

Specification

Alias: none

Argument(s): REAL

Default: 1.e-8

Description

See parent page. The `local_balance_parameter` provides a tolerance for estimating whether the smallest subregion can provide a sufficient decrease to be worth subdividing; the default value is a small value that is suitable for most applications.

7.2.38.4 `max_boxsize_limit`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [max_boxsize_limit](#)

Stopping Criterion based on longest edge of hyperrectangle

Specification

Alias: none

Argument(s): REAL

Default: 0.0

Description

Each subregion considered by DIRECT has a **size**, which corresponds to the longest diagonal of the subregion.

`max_boxsize_limit` specification terminates DIRECT if the size of the largest subregion falls below this threshold.

7.2.38.5 `min_boxsize_limit`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [min_boxsize_limit](#)

Stopping Criterion based on shortest edge of hyperrectangle

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-4

Description

`min_boxsize_limit` is a setting that terminates the optimization when the measure of a hyperrectangle S with $f(c(S)) = f_{\min}$ is less than `min_boxsize_limit`.

Each subregion considered by DIRECT has a **size**, which corresponds to the longest diagonal of the subregion.

`min_boxsize_limit` specification terminates DIRECT if the size of the smallest subregion falls below this threshold.

In practice, this specification is likely to be more effective at limiting DIRECT's search.

7.2.38.6 `constraint_penalty`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [constraint_penalty](#)

Multiplier for the penalty function

Specification

Alias: none

Argument(s): REAL

Default: 1000.0

Description

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

7.2.38.7 `solution_target`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: `-DBL_MAX`

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.38.8 `seed`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.38.9 show_misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [show_misc_options](#)

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

Argument(s): none

Default: no dump of specification options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.38.10 misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [misc_options](#)

Set method options not available through Dakota spec

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.38.11 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.38.12 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.38.13 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.38.14 `scaling`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.38.15 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [coliny_direct](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.39 coliny_ea

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)

Evolutionary Algorithm

Topics

This keyword is related to the topics:

- [package_scolib](#)
- [package_coliny](#)
- [global_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		population_size	Set the population size
	Optional		initialization_type	Specify how to initialize the population
	Optional Optional		fitness_type replacement_type	Select fitness type Select a replacement type for SCOLIB evolutionary algorithm (coliny_ea)
	Optional		crossover_rate	Specify the probability of a crossover event
	Optional		crossover_type	Select a crossover type
	Optional		mutation_rate	Set probability of a mutation
	Optional		mutation_type	Select a mutation type
	Optional		constraint_penalty	Multiplier for the penalty function

	Optional	solution_target	Stopping criteria based on objective function value
	Optional	seed	Seed of the random number generator
	Optional	show_misc_options	Show algorithm parameters not exposed in Dakota input
	Optional	misc_options	Set method options not available through Dakota spec
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

Evolutionary Algorithm

See the page [package_scolib](#) for important information regarding all SCOLIB methods

`coliny_pattern_search` supports concurrency up to the size of the population

The random `seed` control provides a mechanism for making a stochastic optimization repeatable. That is, the use of the same random seed in identical studies will generate identical results. The `population_size` control specifies how many individuals will comprise the EA's population.

The `initialization_type` defines the type of initialization for the population of the EA. There are three types: `simple_random`, `unique_random`, and `flat_file`. `simple_random` creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs. The number of designs is specified by the `population_size`. `unique_random` is the same as `simple_random`, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected. `flat_file` allows the initial population to be read from a flat file. If `flat_file` is specified, a file name must be given.

The `fitness_type` controls how strongly differences in "fitness" (i.e., the objective function) are weighted in the process of selecting "parents" for crossover:

- the `linear_rank` setting uses a linear scaling of probability of selection based on the rank order of each individual's objective function within the population
- the `merit_function` setting uses a proportional scaling of probability of selection based on the relative value of each individual's objective function within the population

The `replacement_type` controls how current populations and newly generated individuals are combined to create a new population. Each of the `replacement_type` selections accepts an integer value, which is referred to below as the `replacement_size`.

- The `random` setting creates a new population using (a) `replacement_size` randomly selected individuals from the current population, and (b) `population_size - replacement_size` individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using `new_solutions_generated`) that are created for each generation (using the selection, crossover, and mutation procedures).
- The `chc` setting creates a new population using (a) the `replacement_size` best individuals from the *combination* of the current population and the newly generated individuals, and (b) `population_size - replacement_size` individuals randomly selected from among the remaining individuals in this combined pool. The `chc` setting is the preferred selection for many engineering problems.
- The `elitist` (default) setting creates a new population using (a) the `replacement_size` best individuals from the current population, (b) and `population_size - replacement_size` individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default `new_solutions_generated` value is set such that the entire set of newly generated individuals will be selected for replacement.

Note that `new_solutions_generated` is not recognized by Dakota as a valid keyword unless `replacement_type` has been specified.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

Theory

The basic steps of an evolutionary algorithm are depicted in Figure 5.2.

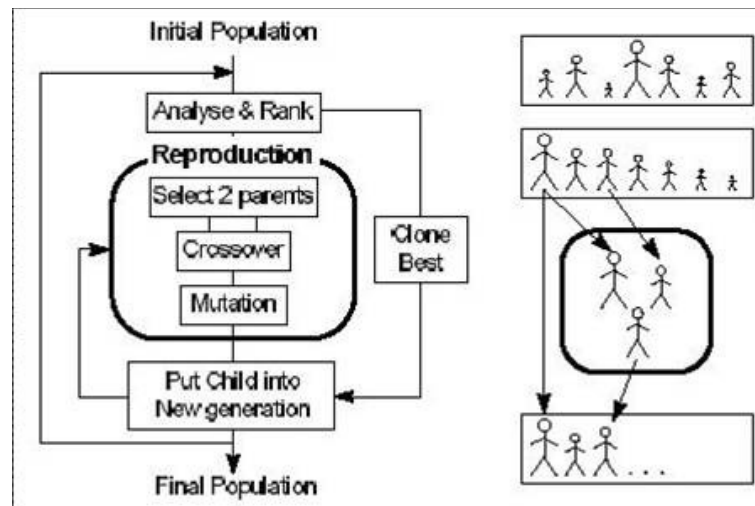


Figure 7.2: Depiction of evolutionary algorithm

They can be enumerated as follows:

1. Select an initial population randomly and perform function evaluations on these individuals
2. Perform selection for parents based on relative fitness
3. Apply crossover and mutation to generate `new_solutions_generated` new individuals from the selected parents
 - Apply crossover with a fixed probability from two selected parents
 - If crossover is applied, apply mutation to the newly generated individual with a fixed probability
 - If crossover is not applied, apply mutation with a fixed probability to a single selected parent
4. Perform function evaluations on the new individuals
5. Perform replacement to determine the new population
6. Return to step 2 and continue the algorithm until convergence criteria are satisfied or iteration limits are exceeded

See Also

These keywords may also be of interest:

- [coliny_beta](#)
- [coliny_direct](#)
- [coliny_pattern_search](#)
- [coliny_cobyla](#)
- [coliny_solis_wets](#)

7.2.39.1 population_size

- [Keywords Area](#)
- [method](#)

- [coliny_ea](#)
- [population_size](#)

Set the population size

Specification

Alias: none

Argument(s): INTEGER

Default: 50

Description

The number of designs in the population is specified by the `population_size`.

7.2.39.2 initialization_type

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [initialization_type](#)

Specify how to initialize the population

Specification

Alias: none

Argument(s): none

Default: `unique_random`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Initialization Type (Group 1)	simple_random	Create random initial solutions
			unique_random	Create random initial solutions, but enforce uniqueness (default)
			flat_file	Read initial solutions from file

Description

The `initialization_type` defines how the initial population is created for the GA. There are three types:

1. `simple_random`
2. `unique_random` (default)

3. flat_file

Setting the size for the `flat_file` initializer has the effect of requiring a minimum number of designs to create. If this minimum number has not been created once the files are all read, the rest are created using the `unique_random` initializer and then the `simple_random` initializer if necessary.

7.2.39.2.1 simple_random

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [initialization_type](#)
- [simple_random](#)

Create random initial solutions

Specification

Alias: none

Argument(s): none

Description

`simple_random` creates initial solutions with random variable values according to a uniform random number distribution. It gives no consideration to any previously generated designs.

7.2.39.2.2 unique_random

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [initialization_type](#)
- [unique_random](#)

Create random initial solutions, but enforce uniqueness (default)

Specification

Alias: none

Argument(s): none

Description

`unique_random` is the same as `simple_random`, except that when a new solution is generated, it is checked against the rest of the solutions. If it duplicates any of them, it is rejected.

7.2.39.2.3 flat_file

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [initialization_type](#)
- [flat_file](#)

Read initial solutions from file

Specification

Alias: none

Argument(s): STRING

Description

flat_file allows the initial population to be read from a flat file. If flat_file is specified, a file name must be given.

7.2.39.3 fitness_type

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [fitness_type](#)

Select fitness type

Specification

Alias: none

Argument(s): none

Default: linear_rank

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Fitness Type (Group 1)	linear_rank	Set selection scaling
			merit_function	Balance goals of reducing objective function and satisfying constraints

Description

The `fitness_type` controls how strongly differences in "fitness" (i.e., the objective function) are weighted in the process of selecting "parents" for crossover. It has two options, `linear_rank` and `merit_function`.

7.2.39.3.1 `linear_rank`

- [Keywords Area](#)
- `method`
- `coliny_ea`
- `fitness_type`
- `linear_rank`

Set selection scaling

Specification

Alias: none

Argument(s): none

Description

The `linear_rank` setting uses a linear scaling of probability of selection based on the rank order of each individual's objective function within the population

7.2.39.3.2 `merit_function`

- [Keywords Area](#)
- `method`
- `coliny_ea`
- `fitness_type`
- `merit_function`

Balance goals of reducing objective function and satisfying constraints

Specification

Alias: none

Argument(s): none

Description

A `merit_function` is a function in constrained optimization that attempts to provide joint progress toward reducing the objective function and satisfying the constraints.

7.2.39.4 replacement_type

- [Keywords Area](#)
- [method](#)
- [colony_ea](#)
- [replacement_type](#)

Select a replacement type for SCOLIB evolutionary algorithm (`colony_ea`)

Specification

Alias: none

Argument(s): none

Default: elitist=1

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Replacement Type (Group 1)	random	Create new population randomly
			chc	Create new population using replacement
			elitist	Use the best designs to form a new population
	Optional		new_solutions_generated	Replace population with individuals chosen from population

Description

The `replacement_type` controls how current populations and newly generated individuals are combined to create a new population. Each of the `replacement_type` selections accepts an associated integer value, which is specified by the `replacement_size`:

The `random` setting creates a new population using (a) `replacement_size` randomly selected individuals from the current population, and (b) `population_size - replacement_size` individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using `new_solutions_generated`) that are created for each generation (using the selection, crossover, and mutation procedures).

The `chc` setting creates a new population using (a) the `replacement_size` best individuals from the combination of the current population and the newly generated individuals, and (b) `population_size - replacement_size` individuals randomly selected from among the remaining individuals in this combined pool. The `chc` setting is the preferred selection for many engineering problems.

The `elitist` (default) setting creates a new population using (a) the `replacement_size` best individuals from the current population, (b) and `population_size - replacement_size` individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default `new_solutions_generated` value is set such that the entire set of newly generated individuals will be selected for replacement.

Note that `new_solutions_generated` is not recognized by Dakota as a valid keyword unless `replacement_type` has been specified.

7.2.39.4.1 random

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [replacement_type](#)
- [random](#)

Create new population randomly

Specification

Alias: none

Argument(s): INTEGER

Description

The `replacement_type` controls how current populations and newly generated individuals are combined to create a new population. Each of the `replacement_type` selections accepts an integer value, which is referred as the `replacement_size`:

The `random` setting creates a new population using:

- `replacement_size` randomly selected individuals from the current population, and
- `population_size - replacement_size` individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using `new_solutions_generated`) that are created for each generation (using the selection, crossover, and mutation procedures).

7.2.39.4.2 chc

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [replacement_type](#)
- [chc](#)

Create new population using replacement

Specification

Alias: none

Argument(s): INTEGER

Description

The `replacement_type` controls how current populations and newly generated individuals are combined to create a new population. Each of the `replacement_type` selections accepts an integer value, which is referred as the `replacement_size`:

The `chc` setting creates a new population using (a) the `replacement_size` best individuals from the *combination* of the current population and the newly generated individuals, and (b) `population_size - replacement_size` individuals randomly selected from among the remaining individuals in this combined pool. The `chc` setting is the preferred selection for many engineering problems.

7.2.39.4.3 elitist

- [Keywords Area](#)
- [method](#)
- [colony_ea](#)
- [replacement_type](#)
- [elitist](#)

Use the best designs to form a new population

Specification

Alias: none

Argument(s): INTEGER

Description

The `elitist` (default) setting creates a new population using (a) the `replacement_size` best individuals from the current population, (b) and `population_size - replacement_size` individuals randomly selected from the newly generated individuals. It is possible in this case to lose a good solution from the newly generated individuals if it is not randomly selected for replacement; however, the default `new_solutions_generated` value is set such that the entire set of newly generated individuals will be selected for replacement.

7.2.39.4.4 new_solutions_generated

- [Keywords Area](#)
- [method](#)
- [colony_ea](#)
- [replacement_type](#)
- [new_solutions_generated](#)

Replace population with individuals chosen from population

Specification

Alias: none

Argument(s): INTEGER

Default: `population_size - replacement_size`

Description

- The `random` setting creates a new population using (a) `replacement_size` randomly selected individuals from the current population, and (b) `population_size - replacement_size` individuals randomly selected from among the newly generated individuals (the number of which is optionally specified using `new_solutions_generated`) that are created for each generation (using the selection, crossover, and mutation procedures).

7.2.39.5 `crossover_rate`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [crossover_rate](#)

Specify the probability of a crossover event

Specification

Alias: none

Argument(s): REAL

Default: 0.8

Description

The `crossover_type` controls what approach is employed for combining parent genetic information to create offspring, and the `crossover_rate` specifies the probability of a crossover operation being performed to generate a new offspring. The SCOLIB EA method supports three forms of crossover, `two_point`, `blend`, and `uniform`, which generate a new individual through combinations of two parent individuals. Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate. Uniform crossover creates offspring through random combination of coordinates from the two parents. Blend crossover generates a new individual randomly along the multidimensional vector connecting the two parents.

7.2.39.6 `crossover_type`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [crossover_type](#)

Select a crossover type

Specification**Alias:** none**Argument(s):** none**Default:** two_point**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Crossover Type (Group 1)	two_point	Combine middle of one parent with end of another
			blend	Random blend of parents
			uniform	Randomly combine coordinates from parents

Description

The `crossover_type` controls what approach is employed for combining parent genetic information to create offspring. The SCOLIB EA method supports three forms of crossover, `two_point`, `blend`, and `uniform`, which generate a new individual through combinations of two parent individuals.

7.2.39.6.1 `two_point`

- [Keywords Area](#)
- [method](#)
- [colony_ea](#)
- [crossover_type](#)
- [two_point](#)

Combine middle of one parent with end of another

Specification

Alias: none

Argument(s): none

Description

Two-point crossover divides each parent into three regions, where offspring are created from the combination of the middle region from one parent and the end regions from the other parent. Since the SCOLIB EA does not utilize bit representations of variable values, the crossover points only occur on coordinate boundaries, never within the bits of a particular coordinate.

7.2.39.6.2 `blend`

- [Keywords Area](#)
- [method](#)
- [colony_ea](#)
- [crossover_type](#)
- [blend](#)

Random blend of parents

Specification

Alias: none

Argument(s): none

Description

`blend` crossover generates a new individual randomly along the multidimensional vector connecting the two parents.

7.2.39.6.3 `uniform`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [crossover_type](#)
- [uniform](#)

Randomly combine coordinates from parents

Specification

Alias: none

Argument(s): none

Description

Uniform crossover creates offspring through random combination of coordinates from the two parents.

7.2.39.7 `mutation_rate`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_rate](#)

Set probability of a mutation

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

The `mutation_rate` controls the probability of mutation being performed on an individual, both for new individuals generated by crossover (if crossover occurs) and for individuals from the existing population. It is the fraction of trial points that are mutated in a given iteration and therefore must be specified to be between 0 and 1.

7.2.39.8 `mutation_type`

- [Keywords Area](#)
- [method](#)
- [colony_ea](#)
- [mutation_type](#)

Select a mutation type

Specification

Alias: none

Argument(s): none

Default: `offset_normal`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Mutation Type (Group 1)	replace_uniform	Replace coordinate with randomly generated value
			offset_normal	Set mutation offset to use a normal distribution
			offset_cauchy	Use a Cauchy distribution for the mutation offset
			offset_uniform	Set mutation offset to use a uniform distribution
	Optional		non_adaptive	Disable self-adaptive mutation

Description

The `mutation_type` controls what approach is employed in randomly modifying continuous design variables within the EA population. Each of the mutation methods generates coordinate-wise changes to individuals, usually by adding a random variable to a given coordinate value (an `offset_*` mutation), but also by replacing a given coordinate value with a random variable (a `replace_*` mutation).

Discrete design variables are always mutated using the `offset_uniform` method.

7.2.39.8.1 `replace_uniform`

- [Keywords Area](#)
- [method](#)

- [coliny_ea](#)
- [mutation_type](#)
- [replace_uniform](#)

Replace coordinate with randomly generated value

Specification

Alias: none

Argument(s): none

Description

The `replace_uniform` mutation type generates a replacement value for a coordinate using a uniformly distributed value over the total range for that coordinate.

7.2.39.8.2 `offset_normal`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_normal](#)

Set mutation offset to use a normal distribution

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mutation_scale	Scales mutation across range of parameter
	Optional		mutation_range	Set uniform offset control for discrete parameters

Description

The `offset_normal` type is an "offset" mutation that adds a 0-mean random variable with a normal uniform distribution to the existing coordinate value. The offset is limited in magnitude by `mutation_scale`.

7.2.39.8.2.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_normal](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.1

Description

The `mutation_scale` specifies a scale factor which scales continuous mutation offsets; this is a fraction of the total range of each dimension, so `mutation_scale` is a relative value between 0 and 1.

7.2.39.8.2.2 `mutation_range`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_normal](#)
- [mutation_range](#)

Set uniform offset control for discrete parameters

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The `mutation_range` is used to control `offset_uniform` mutation used for discrete parameters. The replacement discrete value is the original value plus or minus an integer value up to `mutation_range`.

7.2.39.8.3 `offset_cauchy`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_cauchy](#)

Use a Cauchy distribution for the mutation offset

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mutation_scale	Scales mutation across range of parameter
	Optional		mutation_range	Set uniform offset control for discrete parameters

Description

The `offset_cauchy` type is an "offset" mutation that adds a 0-mean random variable with a cauchy distribution to the existing coordinate value. The offset is limited in magnitude by `mutation_scale`.

7.2.39.8.3.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_cauchy](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.1

Description

The `mutation_scale` specifies a scale factor which scales continuous mutation offsets; this is a fraction of the total range of each dimension, so `mutation_scale` is a relative value between 0 and 1.

7.2.39.8.3.2 `mutation_range`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_cauchy](#)
- [mutation_range](#)

Set uniform offset control for discrete parameters

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The `mutation_range` is used to control `offset_uniform` mutation used for discrete parameters. The replacement discrete value is the original value plus or minus an integer value up to `mutation_range`.

7.2.39.8.4 `offset_uniform`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_uniform](#)

Set mutation offset to use a uniform distribution

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	mutation_scale	Scales mutation across range of parameter
	Optional	mutation_range	Set uniform offset control for discrete parameters

Description

The `offset_uniform` type is an "offset" mutation that adds a 0-mean random variable with a uniform distribution to the existing coordinate value. The offset is limited in magnitude by `mutation_scale`

For discrete design variables, `offset_uniform` is always used, and `mutation_range` controls the magnitude of the mutation.

7.2.39.8.4.1 `mutation_scale`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_uniform](#)
- [mutation_scale](#)

Scales mutation across range of parameter

Specification

Alias: none

Argument(s): REAL

Default: 0.1

Description

The `mutation_scale` specifies a scale factor which scales continuous mutation offsets; this is a fraction of the total range of each dimension, so `mutation_scale` is a relative value between 0 and 1.

7.2.39.8.4.2 `mutation_range`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [offset_uniform](#)
- [mutation_range](#)

Set uniform offset control for discrete parameters

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The `mutation_range` is used to control `offset_uniform` mutation used for discrete parameters. The replacement discrete value is the original value plus or minus an integer value up to `mutation_range`.

7.2.39.8.5 `non_adaptive`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [mutation_type](#)
- [non_adaptive](#)

Disable self-adaptive mutation

Specification

Alias: none

Argument(s): none

Default: Adaptive mutation

Description

The SCOLIB EA method uses self-adaptive mutation, which modifies the mutation scale dynamically. This mechanism is borrowed from EAs like evolution strategies. The `non_adaptive` flag can be used to deactivate the self-adaptation, which may facilitate a more global search.

Note that `non_adaptive` is not recognized by Dakota as a valid keyword unless `mutation_type` has been specified.

7.2.39.9 `constraint_penalty`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [constraint_penalty](#)

Multiplier for the penalty function

Specification

Alias: none

Argument(s): REAL

Description

Most SCOLIB optimizers treat constraints with a simple penalty scheme that adds `constraint_penalty` times the sum of squares of the constraint violations to the objective function. The default value of `constraint_penalty` is 1000.0, except for methods that dynamically adapt their constraint penalty, for which the default value is 1.0.

7.2.39.10 `solution_target`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: `-DBL_MAX`

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.39.11 `seed`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.39.12 `show_misc_options`

- [Keywords Area](#)
- [method](#)
- [colony_ea](#)
- [show_misc_options](#)

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

Argument(s): none

Default: no dump of specification options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.39.13 misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [misc_options](#)

Set method options not available through Dakota spec

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.39.14 max_iterations

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.39.15 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.39.16 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.39.17 `scaling`

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)

- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0, 1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;

- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.39.18 model_pointer

- [Keywords Area](#)
- [method](#)
- [coliny_ea](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.40 coliny_beta

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)

(Experimental) Coliny beta solver

Topics

This keyword is related to the topics:

- [package_scolib](#)
- [package_coliny](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		beta_solver_name	Use an in-development SCOLIB solver
	Optional		solution_target	Stopping criteria based on objective function value
	Optional		seed	Seed of the random number generator
	Optional		show_misc_options	Show algorithm parameters not exposed in Dakota input
	Optional		misc_options	Set method options not available through Dakota spec

	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

This method keyword allows testing of experimental (beta) Coliny (Scolib) optimization solvers during software development. It is intended primarily for developer use. Additional information on Coliny solvers is available at [package_scolib](#).

See Also

These keywords may also be of interest:

- [coliny_direct](#)
- [coliny_pattern_search](#)
- [coliny_cobyala](#)
- [coliny_ea](#)
- [coliny_solis_wets](#)

7.2.40.1 beta_solver_name

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [beta_solver_name](#)

Use an in-development SCOLIB solver

Specification

Alias: none

Argument(s): STRING

Description

This is a means of accessing new methods in SCOLIB before they are exposed through the Dakota interface. Seek help from a Dakota or SCOLIB developer or a Dakota developer.

7.2.40.2 `solution_target`

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [solution_target](#)

Stopping criteria based on objective function value

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: `-DBL_MAX`

Description

`solution_target` is a termination criterion. The algorithm will terminate when the function value falls below `solution_target`.

7.2.40.3 `seed`

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.40.4 show_misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [show_misc_options](#)

Show algorithm parameters not exposed in Dakota input

Specification

Alias: none

Argument(s): none

Default: no dump of specification options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.40.5 misc_options

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [misc_options](#)

Set method options not available through Dakota spec

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

All SCOLIB methods support the `show_misc_options` optional specification which results in a dump of all the allowable method inputs. Note that the information provided by this command refers to optimizer parameters that are internal to SCOLIB, and which may differ from corresponding parameters used by the Dakota interface. The `misc_options` optional specification provides a means for inputting additional settings supported by the SCOLIB methods but which are not currently mapped through the Dakota input specification. Care must be taken in using this specification; they should only be employed by users familiar with the full range of parameter specifications available directly from SCOLIB and understand any differences that exist between those specifications and the ones available through Dakota.

7.2.40.6 max_iterations

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.40.7 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.40.8 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.40.9 scaling

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)

- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0, 1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;

- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.40.10 model_pointer

- [Keywords Area](#)
- [method](#)
- [coliny_beta](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.41 nl2sol

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)

Trust-region method for nonlinear least squares

Topics

This keyword is related to the topics:

- [nonlinear_least_squares](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		function_precision	Specify the maximum precision of the analysis code responses
	Optional		absolute_conv_tol	Absolute convergence tolerance
	Optional		x_conv_tol	X-convergence tolerance
	Optional		singular_conv_tol	Singular convergence tolerance
	Optional		singular_radius	Singular radius
	Optional		false_conv_tol	False convergence tolerance
	Optional		initial_trust_radius	Initial trust region radius
	Optional		covariance	Determine how the final covariance matrix is computed

	Optional	regression_ - diagnostics	Turn on regression diagnostics
	Optional	convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	max_ iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	speculative	Compute speculative gradients
	Optional	max_ function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_ pointer	Identifier for model block to be used by a method

Description

NL2SOL is available as `nl2sol` and addresses unconstrained and bound-constrained least squares problems. It uses a trust-region method (and thus can be viewed as a generalization of the Levenberg-Marquardt algorithm) and adaptively chooses between two Hessian approximations, the Gauss-Newton approximation alone and the Gauss-Newton approximation plus a quasi-Newton approximation to the rest of the Hessian. Even on small-residual problems, the latter Hessian approximation can be useful when the starting guess is far from the solution. On problems that are not over-parameterized (i.e., that do not involve more optimization variables than the data support), NL2SOL usually exhibits fast convergence.

Several internal NL2SOL convergence tolerances are adjusted in response to `function_precision`, which gives the relative precision to which responses are computed.

These tolerances may also be specified explicitly using:

- `convergence_tolerance` (NL2SOL's `rfctol`)
- `x_conv_tol` (NL2SOL's `xctol`)
- `absolute_conv_tol` (NL2SOL's `afctol`)
- `singular_conv_tol` (NL2SOL's `sctol`)
- `false_conv_tol` (NL2SOL's `xftol`)
- `initial_trust_radius` (NL2SOL's `lmax0`)

The internal NL2SOL defaults can be obtained for many of these controls by specifying the value -1. The internal defaults are often functions of machine epsilon (as limited by `function_precision`).

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)
- [Parameter Confidence Intervals](#) (when `num_experiments` equals 1)

Examples

An example of `nl2sol` is given below, and is discussed in the User's Manual.

Note that in this usage of `calibration_terms`, the driver script `rosenbrock`, is returning "residuals", which the `nl2sol` method is attempting to minimize. Another use case is to provide a data file, which Dakota will attempt to match the model responses to. See [calibration_data_file](#). Finally, as of Dakota 6.2, the field data capability may be used with `nl2sol`. That is, the user can specify field simulation data and field experiment data, and Dakota will interpolate and provide the proper residuals for the calibration.

```
# Dakota Input File: rosen_opt_nls.in
environment
  tabular_data
    tabular_data_file = 'rosen_opt_nls.dat'

method
  max_iterations = 100
  convergence_tolerance = 1e-4
  nl2sol

model
  single

variables
  continuous_design = 2
  initial_point      -1.2      1.0
  lower_bounds       -2.0      -2.0
  upper_bounds       2.0       2.0
  descriptors        'x1'      "x2"

interface
  analysis_driver = 'rosenbrock'
  direct

responses
  calibration_terms = 2
  analytic_gradients
  no_hessians
```

Theory

NL2SOL has a variety of internal controls as described in AT&T Bell Labs CS TR 153 (<http://cm.-bell-labs.com/cm/cs/ctr/153.ps.gz>). A number of existing Dakota controls (method independent controls and responses controls) are mapped into these NL2SOL internal controls. In particular, Dakota's `convergence_tolerance`, `max_iterations`, `max_function_evaluations`, and `fd_gradient_step_size` are mapped directly into NL2SOL's `rfctol`, `mxiter`, `mxfcsl`, and `dltfdj` controls, respectively. In addition, Dakota's `fd_hessian_step_size` is mapped into both `delta0` and `dltfdc`, and Dakota's output verbosity is mapped into NL2SOL's `auxprt` and `outlev` (for normal/verbose/debug output, NL2SOL prints initial guess, final solution, solution statistics, non-default values, and changes to the active bound constraint set on every iteration; for quiet output, NL2SOL prints only the initial guess and final solution; and for silent output, NL2SOL output is suppressed).

See Also

These keywords may also be of interest:

- [nlssol_sqp](#)
- [optpp_g_newton](#)
- [field_calibration_terms](#)

7.2.41.1 function_precision

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [function_precision](#)

Specify the maximum precision of the analysis code responses

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

The `function_precision` control provides the algorithm with an estimate of the accuracy to which the problem functions can be computed. This is used to prevent the algorithm from trying to distinguish between function values that differ by less than the inherent error in the calculation.

7.2.41.2 absolute_conv_tol

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [absolute_conv_tol](#)

Absolute convergence tolerance

Specification

Alias: none

Argument(s): REAL

Default: -1. (use NL2SOL internal default)

Description

`absolute_conv_tol` (NL2SOL's `afctol`) is the absolute function convergence tolerance (stop when half the sum of squares is less than `absolute_conv_tol`, which is mainly of interest on zero-residual test problems).

The internal default is a function of machine epsilon (as limited by `function_precision`). The default is selected with a value of -1.

7.2.41.3 `x_conv_tol`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [x_conv_tol](#)

X-convergence tolerance

Specification

Alias: none

Argument(s): REAL

Default: -1. (use NL2SOL internal default)

Description

`x_conv_tol` maps to the internal NL2SOL control `xctol`. It is the X-convergence tolerance (scaled relative accuracy of the solution variables).

The internal default is a function of machine epsilon (as limited by `function_precision`). The default is selected with a value of -1.

7.2.41.4 `singular_conv_tol`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [singular_conv_tol](#)

Singular convergence tolerance

Specification

Alias: none

Argument(s): REAL

Default: -1. (use NL2SOL internal default)

Description

`singular_conv_tol` (NL2SOL's `sctol`) is the singular convergence tolerance, which works in conjunction with `singular_radius` to test for underdetermined least-squares problems (stop when the relative reduction yet possible in the sum of squares appears less than `singular_conv_tol` for steps of scaled length at most `singular_radius`).

The internal default is a function of machine epsilon (as limited by `function_precision`). The default is selected with a value of -1.

7.2.41.5 `singular_radius`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [singular_radius](#)

Singular radius

Specification

Alias: none

Argument(s): REAL

Default: -1. (use NL2SOL internal default of 1)

Description

`singular_radius` works in conjunction with `singular_conv_tol` to test for underdetermined least-squares problems (stop when the relative reduction yet possible in the sum of squares appears less than `singular_conv_tol` for steps of scaled length at most `singular_radius`).

The internal default results in the internal use of steps of length 1. The default is selected with a value of -1.

7.2.41.6 `false_conv_tol`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [false_conv_tol](#)

False convergence tolerance

Specification

Alias: none

Argument(s): REAL

Default: -1. (use NL2SOL internal default)

Description

`false_conv_tol` (NL2SOL's `xftol`) is the false-convergence tolerance (stop with a suspicion of discontinuity when a more favorable stopping test is not satisfied and a step of scaled length at most `false_conv_tol` is not accepted)

The internal default is a function of machine epsilon (as limited by `function_precision`). The default is selected with a value of -1.

7.2.41.7 `initial_trust_radius`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [initial_trust_radius](#)

Initial trust region radius

Specification

Alias: none

Argument(s): REAL

Default: -1. (use NL2SOL internal default of 1)

Description

`initial_trust_radius` specification (NL2SOL's `lmax0`) specifies the initial trust region radius for the algorithm.

The internal default results in the internal use of steps of length 1. The default is selected with a value of -1.

7.2.41.8 `covariance`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [covariance](#)

Determine how the final covariance matrix is computed

Specification

Alias: none

Argument(s): INTEGER

Default: 0 (no covariance)

Description

`covariance` (NL2SOL's `covreq`) specifies whether and how NL2SOL computes a final covariance matrix.

The desired covariance approximation:

- 0 = default = none
- 1 or -1 ==> $\sigma^2 H^{-1} J^T J H^{-1}$
- 2 or -2 ==> $\sigma^2 H^{-1}$
- 3 or -3 ==> $\sigma^2 (J^T J)^{-1}$
- Negative values ==> estimate the final Hessian H by finite differences of function values only (using `fd_hessian_step_size`)
- Positive values ==> differences of gradients (using `fd_hessian_step_size`)

7.2.41.9 regression_diagnostics

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [regression_diagnostics](#)

Turn on regression diagnostics

Specification

Alias: none

Argument(s): none

Default: no regression diagnostics

Description

When `regression_diagnostics` (NL2SOL's `rdreq`) is specified and a positive-definite final Hessian approximation H is computed, NL2SOL computes and prints a regression diagnostic vector RD such that if omitting the i-th observation would cause alpha times the change in the solution that omitting the j-th observation would cause, then $RD[i] = |\alpha| RD[j]$. The finite-difference step-size tolerance affecting H is `fd_step_size` (NL2SOL's `delta0` and `dltfdc`).

7.2.41.10 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.41.11 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.41.12 speculative

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [speculative](#)

Compute speculative gradients

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no speculation

Description

When performing gradient-based optimization in parallel, `speculative` gradients can be selected to address the load imbalance that can occur between gradient evaluation and line search phases. In a typical gradient-based

optimization, the line search phase consists primarily of evaluating the objective function and any constraints at a trial point, and then testing the trial point for a sufficient decrease in the objective function value and/or constraint violation. If a sufficient decrease is not observed, then one or more additional trial points may be attempted sequentially. However, if the trial point is accepted then the line search phase is complete and the gradient evaluation phase begins. By speculating that the gradient information associated with a given line search trial point will be used later, additional coarse grained parallelism can be introduced by computing the gradient information (either by finite difference or analytically) in parallel, at the same time as the line search phase trial-point function values. This balances the total amount of computation to be performed at each design point and allows for efficient utilization of multiple processors. While the total amount of work performed will generally increase (since some speculative gradients will not be used when a trial point is rejected in the line search phase), the run time will usually decrease (since gradient evaluations needed at the start of each new optimization cycle were already performed in parallel during the line search phase). Refer to [13] for additional details. The speculative specification is implemented for the gradient-based optimizers in the DOT, CONMIN, and OPT++ libraries, and it can be used with dakota numerical or analytic gradient selections in the responses specification (refer to [responses](#) gradient section for information on these specifications). It should not be selected with vendor numerical gradients since vendor internal finite difference algorithms have not been modified for this purpose. In full-Newton approaches, the Hessian is also computed speculatively. NPSOL and NLSSOL do not support speculative gradients, as their gradient-based line search in user-supplied gradient mode (dakota numerical or analytic gradients) is a superior approach for load-balanced parallel execution.

The `speculative` specification enables speculative computation of gradient and/or Hessian information, where applicable, for parallel optimization studies. By speculating that the derivative information at the current point will be used later, the complete data set (all available gradient/Hessian information) can be computed on every function evaluation. While some of these computations will be wasted, the positive effects are a consistent parallel load balance and usually shorter wall clock time. The `speculative` specification is applicable only when parallelism in the gradient calculations can be exploited by Dakota (it will be ignored for `vendor numerical` gradients).

7.2.41.13 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.41.14 scaling

- [Keywords Area](#)
- [method](#)
- [nl2sol](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.41.15 `model_pointer`

- [Keywords Area](#)
- [method](#)

- [nl2sol](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'
```

```
method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative
```

```
model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic
```

```
method
  id_method = 'DACE'
```

```
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.42 nonlinear_cg

- [Keywords Area](#)
- [method](#)
- [nonlinear_cg](#)

(Experimental) nonlinear conjugate gradient optimization

Topics

This keyword is related to the topics:

- [local_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		misc_options	Options for nonlinear CG optimizer
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

This method is an incomplete experimental implementation of nonlinear conjugate gradient optimization, a local, gradient-based solver.

7.2.42.1 misc_options

- [Keywords Area](#)
- [method](#)
- [nonlinear_cg](#)
- [misc_options](#)

Options for nonlinear CG optimizer

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

List of miscellaneous string options to pass to the experimental nonlinear CG solver (see NonlinearCGOptimizer.cpp in the Dakota source code for available controls). Includes controls for step sizes, linesearch control, convergence, etc.

7.2.42.2 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [nonlinear_cg](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.42.3 max_iterations

- [Keywords Area](#)
- [method](#)

- [nonlinear_cg](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.42.4 scaling

- [Keywords Area](#)
- [method](#)
- [nonlinear_cg](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from `method`, these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.42.5 model_pointer

- [Keywords Area](#)
- [method](#)
- [nonlinear_cg](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.43 ncsu_direct

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)

Dividing RECTangles method

Topics

This keyword is related to the topics:

- [global_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		solution_target	Specifies a globally optimal value toward which the optimizer should track
	Optional		min_boxsize_limit	Stopping Criterion based on shortest edge of hyperrectangle
	Optional		volume_boxsize_limit	Stopping criterion based on volume of search space
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods

	Optional	max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional	scaling	Turn on scaling for variables, responses, and constraints
	Optional	model_pointer	Identifier for model block to be used by a method

Description

North Carolina State University (NCSU) has an implementation of the DIRECT algorithm (Dividing RECTangles algorithm that is outlined in the SCOLIB method section above). This version is documented in [28] We have found that the NCSU DIRECT implementation works better and is more robust for some problems than `coliny_direct`. Currently, we maintain both versions of DIRECT in Dakota; in the future, we may deprecate one.

The NCSU DIRECT method is selected with `ncsu_direct`. We have tried to maintain consistency between the keywords in SCOLIB and NCSU implementation of DIRECT, but the algorithms have different parameters, so the keywords sometimes have slightly different meaning.

Stopping Criteria

The algorithm stops based on:

1. [max_iterations](#) - number of iterations
2. [max_function_evaluations](#) - number of function evaluations
3. [solution_target](#) and [convergence_tolerance](#)
4. [min_boxsize_limit](#)
5. [volume_boxsize_limit](#)

This method will always strictly respect the number of iterations, but may slightly exceed the number of function evaluations, as it will always explore all sub-rectangles at the current level.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when [objective_functions](#) are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when [calibration_terms](#) are specified)

See Also

These keywords may also be of interest:

- [coliny_direct](#)

7.2.43.1 `solution_target`

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [solution_target](#)

Specifies a globally optimal value toward which the optimizer should track

Specification

Alias: `solution_accuracy`

Argument(s): REAL

Default: 0

Description

The solution target specifies a goal toward which the optimizer should track.

This is used for test problems, when the true global minimum is known (call it `solution_target := fglobal`). Then, the optimization terminates when $100(f_{\min} - f_{\text{global}}) / \max(1, \text{abs}(f_{\text{global}})) < \text{convergence_tolerance}$. The default for `fglobal` is $-1.0e100$ and the default for convergence tolerance is described at [convergence_tolerance](#).

7.2.43.2 `min_boxsize_limit`

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [min_boxsize_limit](#)

Stopping Criterion based on shortest edge of hyperrectangle

Specification

Alias: none

Argument(s): REAL

Default: $1.0e-4$

Description

`min_boxsize_limit` is a setting that terminates the optimization when the measure of a hyperrectangle S with $f(c(S)) = f_{\min}$ is less than `min_boxsize_limit`.

Each subregion considered by DIRECT has a **size**, which corresponds to the longest diagonal of the subregion.

`min_boxsize_limit` specification terminates DIRECT if the size of the smallest subregion falls below this threshold.

In practice, this specification is likely to be more effective at limiting DIRECT's search.

7.2.43.3 volume_boxsize_limit

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [volume_boxsize_limit](#)

Stopping criterion based on volume of search space

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-6

Description

`volume_boxsize_limit` is a setting that terminates the optimization when the volume of a hyperrectangle S with $f(c(S)) = f_{\min}$ is less than `volume_boxsize_limit` percent of the original hyperrectangle. Basically, `volume_boxsize_limit` stops the optimization when the volume of the particular rectangle which has f_{\min} is less than a certain percentage of the whole volume.

7.2.43.4 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.43.5 max_iterations

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.43.6 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.43.7 scaling

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto`, `log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0 \times 10^{\text{DBL_MIN}}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100

responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.43.8 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [ncsu_direct](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.44 genie_opt_darts

- [Keywords Area](#)
- [method](#)
- [genie_opt_darts](#)

Voronoi-based high-dimensional global Lipschitzian optimization

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed	Seed of the random number generator
	Optional		max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

OPT-Darts method is a fast alternative to DIRECT for global Lipschitzian optimization purposes. Instead of hyperrectangular, OPT-Darts decomposes a high-dimensional domain into Voronoi cells, and places samples via stochastic blue noise instead of deterministic cell division.

To refine a cell, OPT-Darts first adds a new sample within it via spoke-dart sampling, then set the conflict radius to the cell's inscribed hypersphere radius, to avoid adding a sample point that is too close to a prior sample, then divide

that cell (and update its neighboring cells) via the approximate Delaunay graph, and use the computed witnesses to decide the next refinement candidate. These two steps replace the corresponding deterministic center-sample and rectangular cell division in DIRECT, respectively.

OPT-Darts is the first exact stochastic Lipschitzian optimization technique that combines the benefits of guaranteed convergence in [Jones et al. 1993] and high dimensional efficiency in [Spall 2005]. Computing blue noise and Voronoi regions has been intractable in high dimensions, and are being done within OPT-Darts using Spoke-Darts.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Calibration](#) (when `calibration_terms` are specified)

7.2.44.1 `seed`

- [Keywords Area](#)
- [method](#)
- [genie_opt_darts](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.44.2 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [genie_opt_darts](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.44.3 scaling

- [Keywords Area](#)
- [method](#)
- [genie_opt_darts](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval $[0,1]$;
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into $[0,1]$.

3. log - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * DBL_MIN$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.44.4 model_pointer

- [Keywords Area](#)
- [method](#)
- [genie_opt_darts](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.45 genie_direct

- [Keywords Area](#)
- [method](#)
- [genie_direct](#)

Classical high-dimensional global Lipschitzian optimization Classical high-dimensional global Lipschitzian optimization

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed	Seed of the random number generator
	Optional		max_function_ - evaluations	Number of function evaluations allowed for optimizers
	Optional		scaling	Turn on scaling for variables, responses, and constraints
	Optional		model_pointer	Identifier for model block to be used by a method

Description

DIRECT (Dividing RECTangles) partitions the domain into hyperrectangles and uses an iterative Lipschitzian optimization approach to search for a global optimal point.

DIRECT begins by scaling the domain into the unit hypercube by adopting a center-sampling strategy. The objective function is evaluated at the midpoint of the domain, where a lower bound is constructed. In one-dimension, the domain is tri-sected and two new center points are sampled. At each iteration (dividing and sampling), DIRECT identifies intervals that contain the best minimal value of the objective function found up to that point. This strategy of selecting and dividing gives DIRECT its performance and convergence properties compared to other deterministic methods.

The classical DIRECT method [Shubert 1972] has two limitations: poor scaling to high dimensions; and relying on a global K; whose exact value is often unknown. The enhanced DIRECT algorithm [Jones et al. 1993] generalizes [Shubert 1972] to higher dimensions and does not require knowledge of the Lipschitz constant.

7.2.45.1 seed

- [Keywords Area](#)
- [method](#)
- [genie_direct](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.45.2 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [genie_direct](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.45.3 `scaling`

- [Keywords Area](#)
- [method](#)
- [genie_direct](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*_scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto, log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value"
  primary_scales = 1 1 100
```

```
responses
  objective_functions 3
  sense "maximize"
  primary_scale_types = "value" "value" "value"
  primary_scales = 1 1 100
```

7.2.45.4 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [genie_direct](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.46 efficient_global

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)

Global Surrogate Based Optimization, a.k.a. EGO

Topics

This keyword is related to the topics:

- [global_optimization_methods](#)
- [surrogate_based_optimization_methods](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_samples	Initial number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional		batch_size	Total batch size in parallel EGO
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Expected improvement convergence tolerance
	Optional		x_conv_tol	x-convergence tolerance
	Optional		gaussian_process	Gaussian Process surrogate model
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations

	Optional	<code>model_pointer</code>	Identifier for model block to be used by a method
--	-----------------	----------------------------	---

Description

The Efficient Global Optimization (EGO) method was first developed by Jones, Schonlau, and Welch [55]. In EGO, a stochastic response surface approximation for the objective function is developed based on some sample points from the "true" simulation.

Note that several major differences exist between our implementation and that of [55]. First, rather than using a branch and bound method to find the point which maximizes the EIF, we use the DIRECT global optimization method.

Second, we support both global optimization and global nonlinear least squares as well as general nonlinear constraints through abstraction and subproblem recasting.

The efficient global method is in prototype form. Currently, we do not expose any specification controls for the underlying Gaussian process model used or for the optimization of the expected improvement function (which is currently performed by the NCSU DIRECT algorithm using its internal defaults).

By default, EGO uses the Surpack GP (Kriging) model, but the Dakota implementation may be selected instead. If `use_derivatives` is specified the GP model will be built using available derivative data (Surpack GP only).

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Best Parameters](#)
- [Best Objective Functions](#) (when `objective_functions` are specified)
- [Best Nonlinear Constraints](#)
- [Calibration](#) (when `calibration_terms` are specified)

Theory

The particular response surface used is a Gaussian process (GP). The GP allows one to calculate the prediction at a new input location as well as the uncertainty associated with that prediction. The key idea in EGO is to maximize the Expected Improvement Function (EIF). The EIF is used to select the location at which a new training point should be added to the Gaussian process model by maximizing the amount of improvement in the objective function that can be expected by adding that point. A point could be expected to produce an improvement in the objective function if its predicted value is better than the current best solution, or if the uncertainty in its prediction is such that the probability of it producing a better solution is high. Because the uncertainty is higher in regions of the design space with few observations, this provides a balance between exploiting areas of the design space that predict good solutions, and exploring areas where more information is needed. EGO trades off this "exploitation vs. exploration." The general procedure for these EGO-type methods is:

- Build an initial Gaussian process model of the objective function
- Find the point that maximizes the EIF. If the EIF value at this point is sufficiently small, stop.
- Evaluate the objective function at the point where the EIF is maximized. Update the Gaussian process model using this new point.
- Return to the previous step.

See Also

These keywords may also be of interest:

- [surrogate_based_local](#)
- [surrogate_based_global](#)

7.2.46.1 `initial_samples`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [initial_samples](#)

Initial number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: $(d+1)(d+2)/2$

Description

The `initial_samples` keyword is used to define the number of initial samples (i.e., randomly chosen sets of variable values) at which to execute a model. The initial samples may later be augmented in an iterative process.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10*\text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N*(\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type random
    initial_samples = 20
    refinement_samples = 5
```

7.2.46.2 `seed`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)

- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.46.3 `batch_size`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [batch_size](#)

Total batch size in parallel EGO

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		exploration	Portion of batch size dedicated to exploration in parallel EGO
	Optional		synchronization	Select how Dakota schedules a batch of concurrent function evaluations in a parallel algorithm

Description

Refinement candidates are generated by an acquisition function such as maximum expected improvement, which balances exploration and exploitation. Refinement candidates can also be generated by purely explorative metrics such as maximum prediction variance. For a specified `batch_size`, `exploration` specifies the subset of this total that will be dedicated to pure exploration of the parameter space.

Default Behavior All of the batch size is devoted to the standard acquisition approach, balancing exploration and exploitation.

Examples

```
method,
  efficient_global
  seed = 1237
  batch_size = 8    # total
  exploration = 2   # 2 out of 8
```

7.2.46.3.1 exploration

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [batch_size](#)
- [exploration](#)

Portion of batch size dedicated to exploration in parallel EGO

Specification

Alias: none

Argument(s): INTEGER

Description

Refinement candidates are generated by an acquisition function such as maximum expected improvement, which balances exploration and exploitation. Refinement candidates can also be generated by purely explorative metrics

such as maximum prediction variance. For a specified `batch_size`, `exploration` specifies the subset of this total that will be dedicated to pure exploration of the parameter space.

Default Behavior All of the batch size is devoted to the standard acquisition approach, balancing exploration and exploitation.

Examples

```
method,
  efficient_global
  seed = 1237
  batch_size = 8   # total
  exploration = 2  # 2 out of 8
```

7.2.46.3.2 synchronization

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [batch_size](#)
- [synchronization](#)

Select how Dakota schedules a batch of concurrent function evaluations in a parallel algorithm

Specification

Alias: none

Argument(s): none

Default: method specific

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Synchronization (Group 1)	Dakota Keyword blocking	Dakota Keyword Description Complete concurrent evaluation of all evaluations within a batch
			nonblocking	A batch of concurrent evaluations are queried for completions and a partial set is returned to the algorithm

Description

The `synchronization` specification can be used to specify the use of either `blocking` or `nonblocking` schedulers, where the former must block until all evaluations are completed and the latter does not block, returning only the partial set that has been completed for a particular call to the scheduler.

7.2.46.3.2.1 blocking

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [batch_size](#)
- [synchronization](#)
- [blocking](#)

Complete concurrent evaluation of all evaluations within a batch

Specification

Alias: none

Argument(s): none

Description

In the `blocking` case, all of the concurrent evaluations in the batch must be completed and all results are returned together to the algorithm.

These runs are reproducible, assuming use of the same seed in cases with random number generation.

7.2.46.3.2.2 nonblocking

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [batch_size](#)
- [synchronization](#)
- [nonblocking](#)

A batch of concurrent evaluations are queried for completions and a partial set is returned to the algorithm

Specification

Alias: none

Argument(s): none

Description

In the `nonblocking` case, the set of running evaluations is queried for completion and a partial set of results is returned to an asynchronous algorithm, which can then generate new evaluations based on these partial results.

In all cases, the algorithm steps will be subject to parallel timing variabilities / race conditions such that these runs will not generally be repeatable regardless of seed specification.

7.2.46.4 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.46.5 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [convergence_tolerance](#)

Expected improvement convergence tolerance

Specification

Alias: none

Argument(s): REAL

Default: 1.e-12

Description

The Efficient Global Optimization method will terminate if the expected improvement is below the specified tolerance for two subsequent iterations.

7.2.46.6 x_conv_tol

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [x_conv_tol](#)

x-convergence tolerance

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-8

Description

The Efficient Global Optimization method will terminate if the relative change in the best decision variables values x is less than the specified tolerance.

7.2.46.7 gaussian_process

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	GP Implementation (Group 1)	surfpack	Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate
			experimental	Use the experimental Gaussian Process surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.46.7.1 surfpack

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	Exports surrogate model in user-specified format(s)

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.46.7.1.1 export_model

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` surpack)
- Artificial neural network (keyword `neural_network`)
- Radial basis Funtions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the surpack executable with an exported model file.

7.2.46.7.1.2 filename_prefix

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [surpack](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: `exported_surrogate`

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.2.46.7.1.3 formats

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	text_archive	Surrogate model plain-text archive file format
	Optional	binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

7.2.46.7.1.4 `text_archive`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.46.7.1.5 binary_archive

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.46.7.2 dakota

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)

- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the `dakota` version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.46.7.3 experimental

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [experimental](#)

Use the experimental Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		options_file	Filename for a YAML file that specifies Gaussian process options

Description

The experimental GP surrogate is an emerging capability intended to supersede the `dakota` and `surfpack` GP implementations by unifying the best features of both. It also includes additional features such as the ability to export the model and then use it from Python.

See Also

These keywords may also be of interest:

- [experimental_gaussian_process](#)

7.2.46.7.3.1 export_model

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.2.46.7.3.2 filename_prefix

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)

- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.2.46.7.3.3 formats

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)

- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See [export_model](#) and the entries for the format selection keywords for further information.

7.2.46.7.3.4 [text_archive](#)

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.46.7.3.5 binary_archive

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.46.7.3.6 options_file

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [gaussian_process](#)
- [experimental](#)
- [options_file](#)

Filename for a YAML file that specifies Gaussian process options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the Gaussian process that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: gp_options.yaml

GP Parameters:
  Sigma Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  scaler name: standardization
  num restarts: 20
  Nugget:
    fixed nugget: 0.0
    estimate nugget: true
  Bounds:
    lower bound: 3.17e-8
    upper bound: 1.0e-2
  gp seed: 42
  standardize response: false
  Trend:
    estimate trend: true
  Options:
    max degree: 2
    reduced basis: false
```

```
p-norm: 1.0
scaler type: none
regression solver type: SVD
verbosity: 1
kernel type: squared exponential
Length-scale Bounds:
  lower bound: 1.0e-2
  upper bound: 1.0e2
verbosity: 1
```

7.2.46.8 use_derivatives

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.46.9 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.46.9.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002      0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.46.9.1.1 header

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.46.9.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.46.9.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.46.9.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.46.9.3 freeform

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.46.9.4 active_only

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.46.10 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
--	--	--	--------------------------	------------------------------

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.46.10.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) annotated.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.46.10.1.1 header

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [export_approx_points_file](#)

- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent [custom_annotated](#)

7.2.46.10.1.2 [eval_id](#)

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no [eval_id](#) column

Description

See description of parent [custom_annotated](#)

7.2.46.10.1.3 [interface_id](#)

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.46.10.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.46.10.3 freeform

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857  0.2601620081  0.759955
0.89991          1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.46.11 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [efficient_global](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.47 surrogate_based_uq

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)

Generic UQ method for constructing and interrogating a surrogate model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		samples_on_ - emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional		sample_type	Selection of sampling strategy
	Optional		rng	Selection of a random number generator
	Optional		probability_ - refinement	Allow refinement of probability and generalized reliability results using importance sampling
	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value

	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 1)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		seed	Seed of the random number generator
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		model_pointer	Identifier for model block to be used by a method

Description

As surrogate models by stochastic expansion migrate into the model specification, this method provides a general-purpose UQ method to interrogate the surrogate for generating statistics.

This method must identify the surrogate of interest through its `model_pointer`, distinguishing it from fully-

integrated method specifications such as [polynomial_chaos](#), [stoch_collocation](#), and [function_train](#) that couple directly with a simulation model (and form the PCE, SC, FT surrogate approximations implicitly prior to using them for generating statistics).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [stoch_collocation](#)
- [function_train](#)

7.2.47.1 `samples_on_emulator`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
  quadrature_order = 2
  samples_on_emulator = 10000
```

7.2.47.2 `sample_type`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Sample Type (Group 1)	Dakota Keyword lhs	Dakota Keyword Description Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.47.2.1 lhs

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.47.2.2 random

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.47.3 rng

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.47.3.1 mt19937

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.47.3.2 rnum2

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.47.4 probability_refinement

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: `sample_refinement`

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Importance Sampling Approach (Group 1)	import	Importance sampling option for probability refinement

		adapt_import	Importance sampling option for probability refinement
		mm_adapt_import	Importance sampling option for probability refinement
	Optional	refinement_samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.47.4.1 import

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.47.4.2 adapt_import

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [probability_refinement](#)

- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.47.4.3 `mm_adapt_import`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.47.4.4 `refinement_samples`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the initial_samples in a sampling design.

7.2.47.5 final_moments

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.

			<code>central</code>	Output central moments and include them within the set of final statistics.
--	--	--	----------------------	---

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.47.5.1 none

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [final_moments](#)

- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

7.2.47.5.2 standard

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.47.5.3 central

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.47.6 response_levels

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_response_levels</code>	Number of values at which to estimate desired statistics for each response
	Optional		<code>compute</code>	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num-`

`_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06

```
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06
```

Level mappings for each response function:

```
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.6000000000e+11  5.5000000000e-01
  4.0000000000e+11  3.8000000000e-01
  4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  6.0000000000e+04  6.1000000000e-01
  6.5000000000e+04  2.9000000000e-01
  7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.5000000000e+05  5.2000000000e-01
  4.0000000000e+05  9.0000000000e-02
  4.5000000000e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.47.6.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.47.6.2 compute

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.47.6.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.47.6.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.47.6.2.3 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.47.6.2.4 system

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.47.6.2.5 series

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.47.6.2.6 `parallel`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.47.7 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds    = 199.3, 474.63
  upper_bounds    = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs       = 3 4
  abscissas       = 5 8 10 .1 .2 .3 .4
  counts          = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs       = 2
  abscissas       = 3 4
  counts          = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
```

```
no_gradients
no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.47.7.1 num_probability_levels

- [Keywords Area](#)
- [method](#)

- [surrogate_based_uq](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page

7.2.47.8 reliability_levels

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.47.8.1 num_reliability_levels

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.47.9 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.47.9.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.47.10 `distribution`

- [Keywords Area](#)
- [method](#)

- [surrogate_based_uq](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.47.10.1 cumulative

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)

- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.47.10.2 complementary

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.47.11 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

	Optional	drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance
--	-----------------	--------------------------------	--

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.47.11.1 `interaction_order`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)

- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.47.11.2 `drop_tolerance`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,  
  sampling  
  sample_type lhs  
  samples = 100  
  variance_based_decomp  
  drop_tolerance = 0.001
```

7.2.47.12 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: diagonal_covariance for response vector > 10; else full_covariance

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.47.13 full_covariance

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.47.14 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.47.14.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)

- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.47.14.1.1 header

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.47.14.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.47.14.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.47.14.2 annotated

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002         0.26         0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.47.14.3 freeform

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.47.14.4 active_only

- [Keywords Area](#)

- [method](#)
- [surrogate_based_uq](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.47.15 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.47.15.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.47.15.1.1 header

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.47.15.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.47.15.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.47.15.2 annotated

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.47.15.3 freeform

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.47.16 seed

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.47.17 `fixed_seed`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The

same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.47.18 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [surrogate_based_uq](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.48 function_train

- [Keywords Area](#)
- [method](#)
- [function_train](#)

UQ method leveraging a functional tensor train surrogate model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates
	Optional		regression_type	Type of solver for forming function train approximations by regression
	Optional		max_solver_ - iterations	Maximum iterations in determining polynomial coefficients
	Optional		max_cross_ - iterations	Maximum number of iterations for cross-approximation during a rank adaptation.
	Optional		solver_tolerance	Convergence tolerance for the optimizer used during the regression solve.

	Optional		response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Required (<i>Choose One</i>)	Collocation Control (Group 1)	collocation_points	Number of collocation points used to estimate expansion coefficients
			collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
	Optional		rounding_tolerance	An accuracy tolerance that is used to guide rounding during rank adaptation.
	Optional		arithmetic_tolerance	A secondary rounding tolerance used for post-processing
	Optional		start_order	(Initial) polynomial order of each univariate function within the functional tensor train.
	Optional		adapt_order	Activate adaptive procedure for determining the best basis order

	Optional	kick_order	increment used when adapting the basis order in function train methods
	Optional	max_order	Maximum polynomial order of each univariate function within the functional tensor train.
	Optional	max_cv_order_candidates	Limit the number of cross-validation candidates for basis order
	Optional	start_rank	The initial rank used for the starting point during a rank adaptation.
	Optional	adapt_rank	Activate adaptive procedure for determining best rank representation
	Optional	kick_rank	The increment in rank employed during each iteration of the rank adaptation.
	Optional	max_rank	Limits the maximum rank that is explored during a rank adaptation.
	Optional	max_cv_rank_candidates	Limit the number of cross-validation candidates for rank
	Optional	samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)

	Optional	sample_type	Selection of sampling strategy
	Optional	rng	Selection of a random number generator
	Optional	probability_-refinement	Allow refinement of probability and generalized reliability results using importance sampling
	Optional	final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional	gen_reliability_-levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions
	Optional	variance_based_-decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

	Optional (<i>Choose One</i>)	Covariance Type (Group 2)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		seed	Seed of the random number generator
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		model_pointer	Identifier for model block to be used by a method

Description

Tensor train decompositions are approximations that exploit low rank structure in an input-output mapping. Refer to the [function_train](#) surrogate model for additional details.

Usage Tips

This method is a self-contained method alternative to the [function_train](#) surrogate model specification, similar to current method specifications for polynomial chaos and stochastic collocation. In particular, this `function_train` method specification directly couples with a simulation model (optionally identified with a `model_pointer`) and an additional `function_train` surrogate model specification is not required as these options have been embedded within the method specification.

Examples

```
method,
  function_train
    start_order = 2
    start_rank = 2  kick_rank = 2  max_rank = 10
    adapt_rank
    solver_tolerance    = 1e-12
    rounding_tolerance  = 1e-12
    convergence_tol     = 1e-6
    collocation_points  = 100
    samples_on_emulator = 100000
    seed = 531
    response_levels = .1 1. 50. 100. 500. 1000.
    variance_based_decomp
```

See Also

These keywords may also be of interest:

- [function_train](#)
- [polynomial_chaos](#)
- [stoch_collocation](#)

7.2.48.1 p_refinement

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		uniform	Refine an expansion uniformly in all dimensions.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized` `dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which

candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.48.1.1 uniform

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Uniform Refinement Approach (Group 1)	increment_start_rank	candidate generation by advancement of starting rank
			increment_start_order	candidate generation by advancement of starting basis order
			increment_max_rank	candidate generation by advancement of maximum rank
			increment_max_order	candidate generation by advancement of maximum basis order

			increment_max_-rank_order	candidate generation by advancement of maximum rank and maximum basis order
--	--	--	---	---

Description

The `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed.

7.2.48.1.1.1 `increment_start_rank`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_start_rank](#)

candidate generation by advancement of starting rank

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `start_rank` to create new refinement candidates. It is typically used in the absence of `adapt_rank` to more rigidly define a refinement sequence.

7.2.48.1.1.2 `increment_start_order`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_start_order](#)

candidate generation by advancement of starting basis order

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `start_order` to create new refinement candidates. It is typically used in the absence of `adapt_order` to more rigidly define a refinement sequence.

7.2.48.1.1.3 `increment_max_rank`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_max_rank](#)

candidate generation by advancement of maximum rank

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `max_rank` to create new refinement candidates. This option requires the use of `adapt_rank`, for which this advancement creates an expanded range for the cross validation (CV) process. Once the CV process saturates (the best rank is less than the max), then no further advancement can occur and the refinement process (for a particular model level/fidelity) halts.

7.2.48.1.1.4 `increment_max_order`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_max_order](#)

candidate generation by advancement of maximum basis order

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `max_order` to create new refinement candidates. This option requires the use of `adapt_order`, for which this advancement creates an expanded range for the cross validation (CV) process. Once the CV process saturates (the best order is less than the max), then no further advancement can occur and the refinement process (for a particular model level/fidelity) halts.

7.2.48.1.1.5 `increment_max_rank_order`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_max_rank_order](#)

candidate generation by advancement of maximum rank and maximum basis order

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of both `max_rank` and `max_order` to create new refinement candidates. This option requires the use of `adapt_rank` and `adapt_order`, for which this advancement creates expanded ranges for the cross validation (CV) process. Once the CV process saturates for one or more of these two dimensions (the best rank is less than the max rank and/or the best order is less than the max order), then no further advancement can occur in that dimension for a particular model level/fidelity.

7.2.48.2 `max_refinement_iterations`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive generalized  
  p_refinement  
  max_refinement_iterations = 20  
  convergence_tol = 1.e-4  
  sparse_grid_level = 1
```

7.2.48.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.48.4 metric_scale

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group scale of refinement metric (Group 1)	Dakota Keyword relative	Dakota Keyword Description relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output**Usage Tips****Additional Discussion**7.2.48.4.1 `relative`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior**Expected Output****Usage Tips****Additional Discussion**7.2.48.4.2 `absolute`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.48.5 regression_type

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [regression_type](#)

Type of solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Regression Type (Group 1)	Dakota Keyword ls	Dakota Keyword Description Use least squares solver for forming function train approximations by regression
			rls2	Use regularized regression solver for forming function train approximations

Description

Function train approximations are formed based on regression for the set of coefficients described at [function_train](#). Solver options include least squares and regularized regression, where the latter penalizes high-order terms to mitigate overfitting.

Default Behavior

The default regression solver is least squares.

7.2.48.5.1 ls

- [Keywords Area](#)

- [method](#)
- [function_train](#)
- [regression_type](#)
- [ls](#)

Use least squares solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Description

Least squares is the standard solver for forming function train approximations based on regression. This option does not include regularization.

7.2.48.5.2 rls2

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [regression_type](#)
- [rls2](#)

Use regularized regression solver for forming function train approximations

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		l2_penalty	Penalty value applied in regularized regression solver for function train approximations

Description

RLS2 is a regularized least squares solver that applies an $L2$ penalty to mitigate overfitting. It includes the option to specify a penalty parameter.

See Also

These keywords may also be of interest:

- [l2_penalty](#)

7.2.48.5.2.1 l2_penalty

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [regression_type](#)
- [rls2](#)
- [l2_penalty](#)

Penalty value applied in regularized regression solver for function train approximations

Specification

Alias: none

Argument(s): REAL

Description

Regularized regression applies a penalty to aspects of the recovered solution (e.g., cardinality, higher-order oscillatory terms, et al.) to mitigate overfitting. This specification provides the scalar penalty parameter (its value is not currently adapted or cross-validated).

Default Behavior

If this penalty parameter is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rls2](#)

7.2.48.6 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.48.7 max_cross_iterations

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [max_cross_iterations](#)

Maximum number of iterations for cross-approximation during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The number of iterations for the cross-approximation algorithm is limited within the `adapt_rank` procedure.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.48.8 solver_tolerance

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [solver_tolerance](#)

Convergence tolerance for the optimizer used during the regression solve.

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

Defines the function value and function gradient tolerance for optimizer used solving the regression problem for the functional tensor train coefficients.

7.2.48.9 response_scaling

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.48.10 tensor_grid

- [Keywords Area](#)
- [method](#)

- [function_train](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with random sample set

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.48.11 collocation_points

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation_ratio-collocation_points_sequence

7.2.48.12 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.48.13 rounding_tolerance

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [rounding_tolerance](#)

An accuracy tolerance that is used to guide rounding during rank adaptation.

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

The rounding tolerance is used to measure how closely one can approximate a functional tensor train representation at a step of the optimization procedure using one of lower ranks. If this tolerance is small, then less compression is achieved but greater accuracy is obtained. If this tolerance is big, greater compression is achieved at the expense of accuracy.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [adapt_rank](#)
- [arithmetic_tolerance](#)

7.2.48.14 arithmetic_tolerance

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [arithmetic_tolerance](#)

A secondary rounding tolerance used for post-processing

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

After forming one or more function train approximations, this tolerance is used for rolling up final results, in particular, combining multilevel/multifidelity expansions and forming products for higher-order moment estimation. It is generally intended as a looser tolerance than `rounding_tolerance`.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rounding_tolerance](#)

7.2.48.15 start_order

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [start_order](#)

(Initial) polynomial order of each univariate function within the functional tensor train.

Specification

Alias: order

Argument(s): INTEGER

Default: 2

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)

Description

This specifies the initial polynomial order of each univariate function that makes up the functional tensor train decomposition. In the case of `adapt_order`, polynomial order starts from this value and is then advanced; its maximum value is limited by `max_order` and increments are controlled by `kick_order`.

Default Behavior

Default is 2.

See Also

These keywords may also be of interest:

- [adapt_order](#)
- [kick_order](#)
- [max_order](#)

7.2.48.15.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [start_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.48.16 `adapt_order`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [adapt_order](#)

Activate adaptive procedure for determining the best basis order

Specification

Alias: none

Argument(s): none

Default: false

Description

This option activates a cross validation procedure to select the best basis order for representing the current set of data. It searches from `start_order` to `max_order` in increments of `kick_order`.

Default Behavior

No cross validation for basis order.

Examples

This example shows specification of an order adaptation starting at order 2, incrementing by 2, and limited at order 6.

```
model,
  id_model = 'FT'
  surrogate global function_train
    start_rank = 4
    adapt_order start_order = 2 kick_order = 2 max_order = 6
    solver_tolerance = 1e-12
    rounding_tolerance = 1e-12
  dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_order](#)
- [kick_order](#)
- [max_order](#)

7.2.48.17 kick_order

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [kick_order](#)

increment used when adapting the basis order in function train methods

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

Used in combination with `adapt_order`, `kick_order` defines the size of the increment that is used during the cross validation process, within the range of `start_order` and `max_order`.

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_order](#)

7.2.48.18 max_order

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [max_order](#)

Maximum polynomial order of each univariate function within the functional tensor train.

Specification

Alias: none

Argument(s): INTEGER

Default: unsigned short max

Description

This specification is inactive for current adaptation options.

In future adaptation options (i.e., cross-approximation adaptation), this will specify the maximum order that can be obtained during adaptation, where polynomial order will start from `start_order` and be limited by `max_order`.

Default Behavior

Max order is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [start_order](#)

7.2.48.19 max_cv_order_candidates

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)

- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.48.20 start_rank

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [start_rank](#)

The initial rank used for the starting point during a rank adaptation.

Specification

Alias: rank

Argument(s): INTEGER

Default: 2

Description

The `adapt_rank` procedure increments the rank on each iteration, starting from the rank defined by `start_rank`

Default Behavior

Default is 2.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.48.21 adapt_rank

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [adapt_rank](#)

Activate adaptive procedure for determining best rank representation

Specification

Alias: none

Argument(s): none

Default: false

Description

The adaptive algorithm proceeds as follows:

1. Start from rank `start_rank` and form an approximation
2. Adapt the current approximation by searching for a solution with lower rank that achieves L2 accuracy within epsilon tolerance of the reference.
3. If a lower rank solution is found with comparable accuracy, then stop. If not, increase the rank by an amount specified by `kick_rank`.
4. Return to step 2 and continue until either `max_rank` is reached or a converged rank (rank less than current reference with comparable accuracy) is found.

Default Behavior

No cross validation for rank.

Examples

This example shows specification of a rank adaptation starting at rank 2, incrementing by 2, and limited at rank 10.

```
model,  
  id_model = 'FT'  
  surrogate global function_train  
    start_order = 5  
    adapt_rank start_rank = 2 kick_rank = 2 max_rank = 10  
    solver_tolerance = 1e-12  
    rounding_tolerance = 1e-12  
    dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_rank](#)
- [kick_rank](#)
- [max_rank](#)

7.2.48.22 kick_rank

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [kick_rank](#)

The increment in rank employed during each iteration of the rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The `adapt_rank` procedure increments the rank on each iteration, using an increment defined by `kick_rank`

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.48.23 max_rank

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [max_rank](#)

Limits the maximum rank that is explored during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: `size_t` max

Description

The `adapt_rank` procedure increments the rank on each iteration, and the maximum value is limited by `max_rank`

Default Behavior

Max rank is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.48.24 max_cv_rank_candidates

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [max_cv_rank_candidates](#)

Limit the number of cross-validation candidates for rank

Specification

Alias: none

Argument(s): INTEGER

Default: size_t max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_rank](#)
- [increment_max_rank_order](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.48.25 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
  quadrature_order = 2
  samples_on_emulator = 10000
```

7.2.48.26 sample_type

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Sample Type (Group 1)	Dakota Keyword	Dakota Keyword Description
			lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.48.26.1 lhs

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.48.26.2 random

- [Keywords Area](#)
- [method](#)

- [function_train](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.48.27 rng

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.48.27.1 [mt19937](#)

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.48.27.2 rnum2

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.48.28 `probability_refinement`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: `sample_refinement`

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group Importance Sampling Approach (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.48.28.1 `import`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.48.28.2 `adapt_import`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [probability_refinement](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.48.28.3 mm_adapt_import

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.48.28.4 refinement_samples

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.48.29 `final_moments`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
```

```

samples = 50 seed = 1234
response_levels = 3.6e+11 1.2e+05 3.5e+05
final_moments central

```

Overriding the default to `none` as follows:

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```

allows associated nested model mappings to be simplified from:

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.

```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.

```

7.2.48.29.1 none

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```

7.2.48.29.2 standard

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.48.29.3 central

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.48.30 response_levels

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'
```

```

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06
4.0000000000e+05	4.5000000000e+05	1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
3.6000000000e+11	5.5000000000e-01		
4.0000000000e+11	3.8000000000e-01		
4.4000000000e+11	2.3000000000e-01		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
6.0000000000e+04	6.1000000000e-01		
6.5000000000e+04	2.9000000000e-01		
7.0000000000e+04	9.0000000000e-02		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
3.5000000000e+05	5.2000000000e-01		
4.0000000000e+05	9.0000000000e-02		
4.5000000000e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.48.30.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.48.30.2 compute

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
```



```

                                3.5e+05 4.0e+05 4.5e+05
compute reliabilities

```

7.2.48.30.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```

method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities

```

7.2.48.30.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)

- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.48.30.2.3 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.48.30.2.4 system

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system

			parallel	Aggregate response statistics assuming a parallel system
--	--	--	--------------------------	--

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.48.30.2.5 series

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.48.30.2.6 parallel

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.48.31 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```

method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		

```

4.0634975300e+11  3.3000000000e-01
5.4196114379e+11  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
4.6431154744e+04  1.0000000000e+00
5.6511827775e+04  8.0000000000e-01
6.1603813790e+04  5.0000000000e-01
7.8702465755e+04  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
2.3796737090e+05  1.0000000000e+00
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.4111498127e+05  0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.48.31.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.48.32 reliability_levels

- [Keywords Area](#)
- [method](#)

- [function_train](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.48.32.1 `num_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: reliability_levels evenly distributed among response functions

Description

See parent page

7.2.48.33 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.48.33.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.48.34 distribution

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Computes statistics according to cumulative functions

			complementary	Computes statistics according to complementary cumulative functions
--	--	--	-------------------------------	---

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.48.34.1 cumulative

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.48.34.2 complementary

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.48.35 variance_based_decomp

- [Keywords Area](#)

- [method](#)
- [function_train](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
    polynomial_chaos # or stoch_collocation
    sparse_grid_level = 3
    variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.48.35.1 interaction_order

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.48.35.2 drop_tolerance

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

7.2.48.36 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.48.37 full_covariance

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.48.38 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.48.38.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
--	-------------------	----------------------	----------------	----------------------------

	Optional	header	Enable header row in custom-annotated tabular file
	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1         0.0002         0.26           0.76
2              0.90009    1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991    1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.48.38.1.1 header

- [Keywords Area](#)
- [method](#)

- [function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.48.38.1.2 `eval_id`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.48.38.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)

- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.48.38.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated header eval_id`

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9         1.1         0.0002          0.26          0.76
2          NO_ID           0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.48.38.3 freeform

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.48.38.4 active_only

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.48.39 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.48.39.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: `annotated` format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.48.39.1.1 header

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.48.39.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.48.39.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.48.39.2 annotated

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.48.39.3 freeform

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.48.40 seed

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.48.41 fixed_seed

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.48.42 model_pointer

- [Keywords Area](#)
- [method](#)
- [function_train](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.49 multifidelity_function_train

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)

Multifidelity uncertainty quantification using function train expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates

	Optional	statistics_mode	type of statistical metric roll-up for multifidelity UQ methods
	Optional	allocation_control	Sample allocation approach for multifidelity expansions
	Optional	discrepancy_emulation	Formulation for emulation of model discrepancies.
	Optional	rounding_tolerance	An accuracy tolerance that is used to guide rounding during rank adaptation.
	Optional	arithmetic_tolerance	A secondary rounding tolerance used for post-processing
	Optional	regression_type	Type of solver for forming function train approximations by regression
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients
	Optional	max_cross_iterations	Maximum number of iterations for cross-approximation during a rank adaptation.
	Optional	solver_tolerance	Convergence tolerance for the optimizer used during the regression solve.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	collocation_points-_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional	collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
	Optional	start_order_-_sequence	Sequence of start orders used in a multi-stage expansion
	Optional	adapt_order	Activate adaptive procedure for determining the best basis order
	Optional	kick_order	increment used when adapting the basis order in function train methods
	Optional	max_order	Maximum polynomial order of each univariate function within the functional tensor train.

	Optional	max_cv_order_- candidates	Limit the number of cross-validation candidates for basis order
	Optional	start_rank_- sequence	Sequence of start ranks used in a multi-stage expansion
	Optional	adapt_rank	Activate adaptive procedure for determining best rank representation
	Optional	kick_rank	The increment in rank employed during each iteration of the rank adaptation.
	Optional	max_rank	Limits the maximum rank that is explored during a rank adaptation.
	Optional	max_cv_rank_- candidates	Limit the number of cross-validation candidates for rank
	Optional	samples_on_- emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional	sample_type	Selection of sampling strategy
	Optional	rng	Selection of a random number generator
	Optional	probability_- refinement	Allow refinement of probability and generalized reliability results using importance sampling

	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 1)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate

	Optional	export_approx_points_file	Output file for surrogate model value evaluations
	Optional	seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional	fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional	model_pointer	Identifier for model block to be used by a method

Description

As described in the [function_train](#) method and the [function_train](#) model, the function train (FT) approximation is a polynomial expansion that exploits low rank structure within the mapping from input random variables to output quantities of interest (QoI). For multilevel and multifidelity function train approximations, we decompose this expansion into several constituent expansions, one per model form or solution control level, where independent function train approximations are constructed for the low-fidelity/coarse resolution model and one or more levels of model discrepancy.

In a three-model case with low-fidelity (L), medium-fidelity (M), and high-fidelity (H) models and an additive discrepancy approach, we can denote this as:

$$Q^H \approx \hat{Q}_{r_L}^L + \hat{\Delta}_{r_{ML}}^{ML} + \hat{\Delta}_{r_{HM}}^{HM}$$

where Δ^{ij} represents a discrepancy expansion computed from $Q^i - Q^j$ and reduced rank representations of these discrepancies may be targeted ($r_{HM} < r_{ML} < r_L$).

In multifidelity approaches, sample allocation for the constituent expansions can be performed with either no, individual, or integrated adaptive refinement as described in [allocation_control](#).

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Integration and Expansion Moments](#) (expansion moments only)
- [Probability Density](#)
- [Level Mappings](#)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

See Also

These keywords may also be of interest:

- [function_train](#)
- [function_train](#)

7.2.49.1 `p_refinement`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		uniform	Refine an expansion uniformly in all dimensions.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized_refinement` controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized_dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.49.1.1 uniform

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Uniform Refinement Approach (Group 1)	increment_start_rank	candidate generation by advancement of starting rank
			increment_start_order	candidate generation by advancement of starting basis order
			increment_max_rank	candidate generation by advancement of maximum rank
			increment_max_order	candidate generation by advancement of maximum basis order
			increment_max_rank_order	candidate generation by advancement of maximum rank and maximum basis order

Description

The `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed.

7.2.49.1.1.1 increment_start_rank

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_start_rank](#)

candidate generation by advancement of starting rank

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `start_rank` to create new refinement candidates. It is typically used in the absence of `adapt_rank` to more rigidly define a refinement sequence.

7.2.49.1.1.2 increment_start_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_start_order](#)

candidate generation by advancement of starting basis order

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `start_order` to create new refinement candidates. It is typically used in the absence of `adapt_order` to more rigidly define a refinement sequence.

7.2.49.1.1.3 `increment_max_rank`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_max_rank](#)

candidate generation by advancement of maximum rank

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `max_rank` to create new refinement candidates. This option requires the use of `adapt_rank`, for which this advancement creates an expanded range for the cross validation (CV) process. Once the CV process saturates (the best rank is less than the max), then no further advancement can occur and the refinement process (for a particular model level/fidelity) halts.

7.2.49.1.1.4 `increment_max_order`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_max_order](#)

candidate generation by advancement of maximum basis order

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of `max_order` to create new refinement candidates. This option requires the use of `adapt_order`, for which this advancement creates an expanded range for the cross validation (CV) process. Once the CV process saturates (the best order is less than the max), then no further advancement can occur and the refinement process (for a particular model level/fidelity) halts.

7.2.49.1.1.5 increment_max_rank_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [p_refinement](#)
- [uniform](#)
- [increment_max_rank_order](#)

candidate generation by advancement of maximum rank and maximum basis order

Specification

Alias: none

Argument(s): none

Description

Within adaptive refinement processes, this specification selects the advancement of both `max_rank` and `max_order` to create new refinement candidates. This option requires the use of `adapt_rank` and `adapt_order`, for which this advancement creates expanded ranges for the cross validation (CV) process. Once the CV process saturates for one or more of these two dimensions (the best rank is less than the max rank and/or the best order is less than the max order), then no further advancement can occur in that dimension for a particular model level/fidelity.

7.2.49.2 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive_generalized  
  p_refinement  
    max_refinement_iterations = 20  
    convergence_tol = 1.e-4  
  sparse_grid_level = 1
```

7.2.49.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)

- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.49.4 metric_scale

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group scale of refinement metric (Group 1)	Dakota Keyword relative	Dakota Keyword Description relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output

Usage Tips

Additional Discussion

7.2.49.4.1 relative

- [Keywords Area](#)
- [method](#)

- [multifidelity_function_train](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.49.4.2 absolute

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.49.5 `statistics_mode`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [statistics_mode](#)

type of statistical metric roll-up for multifidelity UQ methods

Specification

Alias: none

Argument(s): none

Default: DEFAULT_EXPANSION_STATS

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics mode for refinement metric (Group 1)	Dakota Keyword active	Dakota Keyword Description use statistics for the active expansion within multifidelity refinement
			combined	use statistics for the combined expansion within multifidelity refinement

Description

When performing adaptive refinement in multifidelity UQ methods, one can adapt the expansion for a model level/fidelity based on its local statistics ([active](#)) or based on the statistics for the aggregate expansion ([combined](#)). The latter case may incur additional computational overhead, but ensures that the refinement occurs in the aggregate multifidelity context.

Default Behavior

Default for multifidelity methods is `combined` statistics.

Expected Output**Usage Tips****Additional Discussion**7.2.49.5.1 `active`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [statistics_mode](#)

- [active](#)

use statistics for the active expansion within multifidelity refinement

Specification

Alias: none

Argument(s): none

Description

When performing adaptive refinement in multifidelity UQ methods, this option adapts based on the statistics for the active model level/fidelity. This approach avoids some computational overhead in some cases by avoiding expansion roll-up.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.49.5.2 [combined](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [statistics_mode](#)
- [combined](#)

use statistics for the combined expansion within multifidelity refinement

Specification

Alias: none

Argument(s): none

Description

When performing adaptive refinement in multifidelity UQ methods, this option adapts based on the statistics for the aggregate multifidelity expansion. This approach may incur additional computational overhead, but ensures that the refinement occurs in the aggregate multifidelity context.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.49.6 allocation_control

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity function train

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.49.6.1 greedy

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity function train

Specification

Alias: none

Argument(s): none

Description

Multifidelity function train supports greedy refinement strategies based on regression approaches for computing expansion coefficients. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within a unified competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any $z/p/\beta/\beta^*$ level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity function train starts from a rank-two order-two reference expansion for each level, with twice as many samples as regression coefficients, and generates candidate refinements for each level that are competed in an integrated greedy competition. The number of new samples for the incremented candidate is determined from the collocation ratio times the regression size (which may either be fixed or adapted in the case of `adapt_rank`). In this example, the number of candidates for each level is limited to one uniform refinement of the current expansion, and uniform refinement currently involves an advancement in the basis order for all approximation cores in combination with a rank adaptation between two and ten, incrementing in steps of two.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_function_train
    allocation_control greedy
    p_refinement uniform
      start_rank_sequence = 2 2 2 2 2
      adapt_rank kick_rank = 2 max_rank = 10
      start_order_sequence = 2 2 2 2 2 max_order = 10
      collocation_ratio = 2. seed = 160415
      convergence_tolerance = 1.e-2
      max_refinement_iterations = 5
```

7.2.49.7 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Discrepancy Emulation Approach (Group 1)	distinct	Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.49.7.1 `distinct`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.49.7.2 recursive

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.49.8 rounding_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [rounding_tolerance](#)

An accuracy tolerance that is used to guide rounding during rank adaptation.

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

The rounding tolerance is used to measure how closely one can approximate a functional tensor train representation at a step of the optimization procedure using one of lower ranks. If this tolerance is small, then less compression is achieved but greater accuracy is obtained. If this tolerance is big, greater compression is achieved at the expense of accuracy.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [adapt_rank](#)
- [arithmetic_tolerance](#)

7.2.49.9 arithmetic_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [arithmetic_tolerance](#)

A secondary rounding tolerance used for post-processing

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

After forming one or more function train approximations, this tolerance is used for rolling up final results, in particular, combining multilevel/multifidelity expansions and forming products for higher-order moment estimation. It is generally intended as a looser tolerance than `rounding_tolerance`.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rounding_tolerance](#)

7.2.49.10 regression_type

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [regression_type](#)

Type of solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Regression Type (Group 1)	Dakota Keyword ls	Dakota Keyword Description Use least squares solver for forming function train approximations by regression
			rls2	Use regularized regression solver for forming function train approximations

Description

Function train approximations are formed based on regression for the set of coefficients described at [function_train](#). Solver options include least squares and regularized regression, where the latter penalizes high-order terms to mitigate overfitting.

Default Behavior

The default regression solver is least squares.

7.2.49.10.1 ls

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [regression_type](#)
- [ls](#)

Use least squares solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Description

Least squares is the standard solver for forming function train approximations based on regression. This option does not include regularization.

7.2.49.10.2 rls2

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [regression_type](#)
- [rls2](#)

Use regularized regression solver for forming function train approximations

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		l2_penalty	Penalty value applied in regularized regression solver for function train approximations

Description

RLS2 is a regularized least squares solver that applies an $L2$ penalty to mitigate overfitting. It includes the option to specify a penalty parameter.

See Also

These keywords may also be of interest:

- [l2_penalty](#)

7.2.49.10.2.1 l2_penalty

- [Keywords Area](#)
- [method](#)

- [multifidelity_function_train](#)
- [regression_type](#)
- [rls2](#)
- [l2_penalty](#)

Penalty value applied in regularized regression solver for function train approximations

Specification

Alias: none

Argument(s): REAL

Description

Regularized regression applies a penalty to aspects of the recovered solution (e.g., cardinality, higher-order oscillatory terms, et al.) to mitigate overfitting. This specification provides the scalar penalty parameter (its value is not currently adapted or cross-validated).

Default Behavior

If this penalty parameter is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rls2](#)

7.2.49.11 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.49.12 max_cross_iterations

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [max_cross_iterations](#)

Maximum number of iterations for cross-approximation during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The number of iterations for the cross-approximation algorithm is limited within the `adapt_rank` procedure.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.49.13 solver_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [solver_tolerance](#)

Convergence tolerance for the optimizer used during the regression solve.

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

Defines the function value and function gradient tolerance for optimizer used solving the regression problem for the functional tensor train coefficients.

7.2.49.14 `response_scaling`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.49.15 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with random sample set

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.49.16 collocation_points_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.49.17 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Default: 0.

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.49.18 `start_order_sequence`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [start_order_sequence](#)

Sequence of start orders used in a multi-stage expansion

Specification

Alias: `order_sequence`

Argument(s): INTEGERLIST

Default: 2

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)

Description

Each level entry of the `start_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support start order sequences include multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [start_order](#)

See Also

These keywords may also be of interest:

- [start_order](#)

7.2.49.18.1 dimension_preference

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [start_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.49.19 adapt_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [adapt_order](#)

Activate adaptive procedure for determining the best basis order

Specification

Alias: none

Argument(s): none

Default: false

Description

This option activates a cross validation procedure to select the best basis order for representing the current set of data. It searches from `start_order` to `max_order` in increments of `kick_order`.

Default Behavior

No cross validation for basis order.

Examples

This example shows specification of an order adaptation starting at order 2, incrementing by 2, and limited at order 6.

```
model,
  id_model = 'FT'
  surrogate global function_train
    start_rank = 4
    adapt_order start_order = 2 kick_order = 2 max_order = 6
    solver_tolerance = 1e-12
    rounding_tolerance = 1e-12
  dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_order](#)
- [kick_order](#)
- [max_order](#)

7.2.49.20 kick_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [kick_order](#)

increment used when adapting the basis order in function train methods

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

Used in combination with `adapt_order`, `kick_order` defines the size of the increment that is used during the cross validation process, within the range of `start_order` and `max_order`.

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_order](#)

7.2.49.21 max_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [max_order](#)

Maximum polynomial order of each univariate function within the functional tensor train.

Specification

Alias: none

Argument(s): INTEGER

Default: unsigned short max

Description

This specification is inactive for curren adaptation options.

In future adaptation options (i.e., cross-approximation adaptation), this will specify the maximum order that can be obtained during adaptation, where polynomial order will start from `start_order` and be limited by `max_order`.

Default Behavior

Max order is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [start_order](#)

7.2.49.22 max_cv_order_candidates

- [Keywords Area](#)
- [method](#)

- [multifidelity_function_train](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.49.23 start_rank_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [start_rank_sequence](#)

Sequence of start ranks used in a multi-stage expansion

Specification

Alias: rank_sequence

Argument(s): INTEGERLIST

Default: 2

Description

Each level entry of the `start_rank_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support start rank sequences include multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [start_rank](#)

See Also

These keywords may also be of interest:

- [start_rank](#)

7.2.49.24 `adapt_rank`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [adapt_rank](#)

Activate adaptive procedure for determining best rank representation

Specification

Alias: none

Argument(s): none

Default: false

Description

The adaptive algorithm proceeds as follows:

1. Start from rank `start_rank` and form an approximation
2. Adapt the current approximation by searching for a solution with lower rank that achieves L2 accuracy within epsilon tolerance of the reference.
3. If a lower rank solution is found with comparable accuracy, then stop. If not, increase the rank by an amount specified by `kick_rank`.
4. Return to step 2 and continue until either `max_rank` is reached or a converged rank (rank less than current reference with comparable accuracy) is found.

Default Behavior

No cross validation for rank.

Examples

This example shows specification of a rank adaptation starting at rank 2, incrementing by 2, and limited at rank 10.

```
model,
  id_model = 'FT'
  surrogate global function_train
  start_order = 5
  adapt_rank start_rank = 2 kick_rank = 2 max_rank = 10
  solver_tolerance = 1e-12
  rounding_tolerance = 1e-12
  dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_rank](#)
- [kick_rank](#)
- [max_rank](#)

7.2.49.25 kick_rank

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [kick_rank](#)

The increment in rank employed during each iteration of the rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The `adapt_rank` procedure increments the rank on each iteration, using an increment defined by `kick_rank`

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.49.26 `max_rank`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [max_rank](#)

Limits the maximum rank that is explored during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: `size_t` max

Description

The `adapt_rank` procedure increments the rank on each iteration, and the maximum value is limited by `max_rank`

Default Behavior

Max rank is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.49.27 `max_cv_rank_candidates`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [max_cv_rank_candidates](#)

Limit the number of cross-validation candidates for rank

Specification

Alias: none

Argument(s): INTEGER

Default: `size_t` max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_rank](#)
- [increment_max_rank_order](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.49.28 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.49.29 `sample_type`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Sample Type (Group 1)	Dakota Keyword lhs	Dakota Keyword Description Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.49.29.1 `lhs`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)

- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.49.29.2 random

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.49.30 rng

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (`mt19937`)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.49.30.1 mt19937

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.49.30.2 rnum2

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.49.31 probability_refinement

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: sample_refinement

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Importance Sampling Approach (Group 1)	import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.49.31.1 import

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.49.31.2 `adapt_import`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [probability_refinement](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.49.31.3 `mm_adapt_import`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more

probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.49.31.4 refinement_samples

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.49.32 final_moments

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required(Choose One)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.49.32.1 none

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,  
  sampling  
    samples = 50 seed = 1234  
    response_levels = 3.6e+11 1.2e+05 3.5e+05  
    final_moments none
```

7.2.49.32.2 standard

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.49.32.3 central

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.49.33 response_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_response_levels</code>	Number of values at which to estimate desired statistics for each response
	Optional		<code>compute</code>	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num-`

`_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06


```
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06
```

Level mappings for each response function:

```
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.6000000000e+11  5.5000000000e-01
  4.0000000000e+11  3.8000000000e-01
  4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  6.0000000000e+04  6.1000000000e-01
  6.5000000000e+04  2.9000000000e-01
  7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.5000000000e+05  5.2000000000e-01
  4.0000000000e+05  9.0000000000e-02
  4.5000000000e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.49.33.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.49.33.2 compute

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.49.33.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.49.33.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.49.33.2.3 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.49.33.2.4 system

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.49.33.2.5 series

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.49.33.2.6 `parallel`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.49.34 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which probability_levels correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
```



```
no_gradients
no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.49.34.1 num_probability_levels

- [Keywords Area](#)
- [method](#)

- [multifidelity_function_train](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.49.35 `reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.49.35.1 num_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.49.36 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.49.36.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.49.37 `distribution`

- [Keywords Area](#)
- [method](#)

- [multifidelity_function_train](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.49.37.1 cumulative

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)

- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.49.37.2 complementary

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.49.38 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
    polynomial_chaos # or stoch_collocation
    sparse_grid_level = 3
    variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.49.38.1 `interaction_order`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.49.38.2 `drop_tolerance`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
```

```
sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

7.2.49.39 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: diagonal_covariance for response vector > 10; else full_covariance

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.49.40 full_covariance

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.49.41 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.49.41.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)

- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.49.41.1.1 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.49.41.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.49.41.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.49.41.2 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.49.41.3 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.49.41.4 active_only

- [Keywords Area](#)

- [method](#)
- [multifidelity_function_train](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.49.42 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.49.42.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009        1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991        1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.49.42.1.1 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.49.42.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.49.42.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.49.42.2 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.49.42.3 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.49.43 seed_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For L levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 100 50 40 30 20
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.49.44 `fixed_seed`

- [Keywords Area](#)

- [method](#)
- [multifidelity_function_train](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.49.45 model_pointer

- [Keywords Area](#)
- [method](#)
- [multifidelity_function_train](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.50 multilevel_function_train

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)

Multilevel uncertainty quantification using function train expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		allocation_control	Sample allocation approach for multilevel expansions
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates
	Optional		discrepancy_emulation	Formulation for emulation of model discrepancies.

	Optional	rounding_tolerance	An accuracy tolerance that is used to guide rounding during rank adaptation.
	Optional	arithmetic_tolerance	A secondary rounding tolerance used for post-processing
	Optional	regression_type	Type of solver for forming function train approximations by regression
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients
	Optional	max_cross_iterations	Maximum number of iterations for cross-approximation during a rank adaptation.
	Optional	solver_tolerance	Convergence tolerance for the optimizer used during the regression solve.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

	Optional	collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional	collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
	Optional	start_order_ - sequence	Sequence of start orders used in a multi-stage expansion
	Optional	adapt_order	Activate adaptive procedure for determining the best basis order
	Optional	kick_order	increment used when adapting the basis order in function train methods
	Optional	max_order	Maximum polynomial order of each univariate function within the functional tensor train.
	Optional	max_cv_order_ - candidates	Limit the number of cross-validation candidates for basis order
	Optional	start_rank_ - sequence	Sequence of start ranks used in a multi-stage expansion
	Optional	adapt_rank	Activate adaptive procedure for determining best rank representation

	Optional	kick_rank	The increment in rank employed during each iteration of the rank adaptation.
	Optional	max_rank	Limits the maximum rank that is explored during a rank adaptation.
	Optional	max_cv_rank_- candidates	Limit the number of cross-validation candidates for rank
	Optional	samples_on_- emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional	sample_type	Selection of sampling strategy
	Optional	rng	Selection of a random number generator
	Optional	probability_- refinement	Allow refinement of probability and generalized reliability results using importance sampling
	Optional	final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value

	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 1)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

As described in the [function_train](#) method and the [function_train](#) model, the function train (FT) approximation is a polynomial expansion that exploits low rank structure within the mapping from input random variables to output quantities of interest (QoI). For multilevel and multifidelity function train approximations, we decompose this expansion into several constituent expansions, one per model form or solution control level, where independent function train approximations are constructed for the low-fidelity/coarse resolution model and one or more levels of model discrepancy.

In a three-model case with low-fidelity (L), medium-fidelity (M), and high-fidelity (H) models and an additive discrepancy approach, we can denote this as:

$$Q^H \approx \hat{Q}_{r_L}^L + \hat{\Delta}_{r_{ML}}^{ML} + \hat{\Delta}_{r_{HM}}^{HM}$$

where Δ^{ij} represents a discrepancy expansion computed from $Q^i - Q^j$ and reduced rank representations of these discrepancies may be targeted ($r_{HM} < r_{ML} < r_L$).

In multilevel approaches, sample allocation for the constituent expansions is performed as described in [allocation_control](#).

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Integration and Expansion Moments](#) (expansion moments only)
- [Probability Density](#)
- [Level Mappings](#)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

See Also

These keywords may also be of interest:

- [function_train](#)
- [function_train](#)

7.2.50.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.50.2 allocation_control

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [allocation_control](#)

Sample allocation approach for multilevel expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required (<i>Choose One</i>)	Multilevel Sample Allocation Control (Group 1)	estimator_variance	Variance of mean estimator within multilevel polynomial chaos
			rank_sampling	Sample allocation based on rank sampling within multilevel function train

Description

Multilevel expansions, including regression-based polynomial chaos expansion (PCE) and function train (FT) expansions, require a sample allocation strategy. Three options are currently available:

- allocation based on assuming a convergence rate for the estimator variance (for regression PCE)
- restricted isometry property (RIP) sampling (for regression PCE via compressed sensing)
- rank sampling (for FT)

Default Behavior

Current defaults, when `allocation_control` is not specified, are estimator variance for PCE and rank sampling for FT.

7.2.50.2.1 `estimator_variance`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [allocation_control](#)
- [estimator_variance](#)

Variance of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		estimator_rate	Rate of convergence of mean estimator within multilevel polynomial chaos

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ , with default values that may be overridden as part of this specification block.

This approach is supported for regression-based PCE approaches (over-determined least squares, under-determined compressed sensing, or orthogonal least interpolation).

In practice, it can be challenging to estimate a smooth convergence rate for estimator variance in the presence of abrupt transitions in the quality of sparse recoveries. As a result, sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the convergence of the estimator variance, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```
method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
    orthogonal_matching_pursuit
    expansion_order_sequence = 2
    pilot_samples = 10
    collocation_ratio = .9
    allocation_control
      estimator_variance estimator_rate = 2.5
    seed = 1237
    convergence_tolerance = .01
```

7.2.50.2.1.1 estimator_rate

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [allocation_control](#)
- [estimator_variance](#)
- [estimator_rate](#)

Rate of convergence of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): REAL

Default: 2

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ .

The default values are $\gamma = 1$ and $\kappa = 2$ (adopts a more aggressive sample profile by assuming a faster convergence rate than Monte Carlo). This advanced specification option allows to user to specify κ , overriding the default.

7.2.50.2.2 rank_sampling

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [allocation_control](#)
- [rank_sampling](#)

Sample allocation based on rank sampling within multilevel function train

Specification

Alias: none

Argument(s): none

Description

Multilevel function train based on regression may allocate the number of samples per level based on the collocation ratio times the regression size. The regression size is determined by the rank per core and the basis order per dimension as described at [function_train](#), where these ranks and orders may be either user-specified values (for initial sample allocation), incremented values (for external adaptation by Dakota), or recovered values (in the case of internal C3 adaptation using `adapt_rank`).

The adaptive algorithm starts from a pilot sample, shapes the profile based on the regression size computed from the current orders and recovered ranks, and iterates until convergence.

This capability is **experimental** and under active development.

Examples

This example starts with rank-two order-two initial expansion for each level, with twice as many samples as regression coefficients. As the recovered rank is updated for each level, as dictated by the internal `adapt_rank` approach, the number of samples is incremented as needed in order to synchronize with the specified collocation ratio. In this case, the basis order is fixed and only the ranks and associated samples are updated for each level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_function_train
    allocation_control rank_sampling
  p_refinement uniform
    start_rank_sequence = 2 2 2 2 2
    adapt_rank kick_rank = 2 max_rank = 10
    start_order_sequence = 2 2 2 2 2
    collocation_ratio = 2. seed = 160415
    convergence_tolerance = 1.e-2
    max_refinement_iterations = 5
```

7.2.50.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.50.4 `metric_scale`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group scale of refinement metric (Group 1)	Dakota Keyword relative	Dakota Keyword Description relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output

Usage Tips**Additional Discussion**7.2.50.4.1 `relative`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior**Expected Output****Usage Tips****Additional Discussion**7.2.50.4.2 `absolute`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.50.5 `discrepancy_emulation`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: `distinct`

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword <code>distinct</code>	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			<code>recursive</code>	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting

for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.50.5.1 distinct

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.50.5.2 recursive

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.50.6 `rounding_tolerance`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [rounding_tolerance](#)

An accuracy tolerance that is used to guide rounding during rank adaptation.

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

The rounding tolerance is used to measure how closely one can approximate a functional tensor train representation at a step of the optimization procedure using one of lower ranks. If this tolerance is small, then less compression is achieved but greater accuracy is obtained. If this tolerance is big, greater compression is achieved at the expense of accuracy.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [adapt_rank](#)
- [arithmetic_tolerance](#)

7.2.50.7 arithmetic_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [arithmetic_tolerance](#)

A secondary rounding tolerance used for post-processing

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

After forming one or more function train approximations, this tolerance is used for rolling up final results, in particular, combining multilevel/multifidelity expansions and forming products for higher-order moment estimation. It is generally intended as a looser tolerance than `rounding_tolerance`.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rounding_tolerance](#)

7.2.50.8 regression_type

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [regression_type](#)

Type of solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Regression Type (Group 1)	ls	Use least squares solver for forming function train approximations by regression
			rls2	Use regularized regression solver for forming function train approximations

Description

Function train approximations are formed based on regression for the set of coefficients described at [function_train](#). Solver options include least squares and regularized regression, where the latter penalizes high-order terms to mitigate overfitting.

Default Behavior

The default regression solver is least squares.

7.2.50.8.1 ls

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [regression_type](#)
- [ls](#)

Use least squares solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Description

Least squares is the standard solver for forming function train approximations based on regression. This option does not include regularization.

7.2.50.8.2 rls2

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [regression_type](#)
- [rls2](#)

Use regularized regression solver for forming function train approximations

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		l2_penalty	Penalty value applied in regularized regression solver for function train approximations

Description

RLS2 is a regularized least squares solver that applies an $L2$ penalty to mitigate overfitting. It includes the option to specify a penalty parameter.

See Also

These keywords may also be of interest:

- [l2_penalty](#)

7.2.50.8.2.1 [l2_penalty](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [regression_type](#)
- [rls2](#)
- [l2_penalty](#)

Penalty value applied in regularized regression solver for function train approximations

Specification

Alias: none

Argument(s): REAL

Description

Regularized regression applies a penalty to aspects of the recovered solution (e.g., cardinality, higher-order oscillatory terms, et al.) to mitigate overfitting. This specification provides the scalar penalty parameter (its value is not currently adapted or cross-validated).

Default Behavior

If this penalty parameter is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rls2](#)

7.2.50.9 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.50.10 max_cross_iterations

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [max_cross_iterations](#)

Maximum number of iterations for cross-approximation during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The number of iterations for the cross-approximation algorithm is limited within the `adapt_rank` procedure.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.50.11 solver_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [solver_tolerance](#)

Convergence tolerance for the optimizer used during the regression solve.

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

Defines the function value and function gradient tolerance for optimizer used solving the regression problem for the functional tensor train coefficients.

7.2.50.12 response_scaling

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.50.13 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with random sample set

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.50.14 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.50.15 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Default: 0.

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.50.16 `start_order_sequence`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [start_order_sequence](#)

Sequence of start orders used in a multi-stage expansion

Specification

Alias: `order_sequence`

Argument(s): INTEGERLIST

Default: 2

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)

Description

Each level entry of the `start_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support start order sequences include multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [start_order](#)

See Also

These keywords may also be of interest:

- [start_order](#)

7.2.50.16.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [start_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.50.17 `adapt_order`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [adapt_order](#)

Activate adaptive procedure for determining the best basis order

Specification

Alias: none

Argument(s): none

Default: false

Description

This option activates a cross validation procedure to select the best basis order for representing the current set of data. It searches from `start_order` to `max_order` in increments of `kick_order`.

Default Behavior

No cross validation for basis order.

Examples

This example shows specification of an order adaptation starting at order 2, incrementing by 2, and limited at order 6.

```
model,
  id_model = 'FT'
  surrogate global function_train
  start_rank = 4
  adapt_order start_order = 2 kick_order = 2 max_order = 6
  solver_tolerance = 1e-12
  rounding_tolerance = 1e-12
  dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_order](#)
- [kick_order](#)
- [max_order](#)

7.2.50.18 kick_order

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [kick_order](#)

increment used when adapting the basis order in function train methods

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

Used in combination with `adapt_order`, `kick_order` defines the size of the increment that is used during the cross validation process, within the range of `start_order` and `max_order`.

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_order](#)

7.2.50.19 `max_order`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [max_order](#)

Maximum polynomial order of each univariate function within the functional tensor train.

Specification

Alias: none

Argument(s): INTEGER

Default: unsigned short max

Description

This specification is inactive for current adaptation options.

In future adaptation options (i.e., cross-approximation adaptation), this will specify the maximum order that can be obtained during adaptation, where polynomial order will start from `start_order` and be limited by `max_order`.

Default Behavior

Max order is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [start_order](#)

7.2.50.20 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.50.21 `start_rank_sequence`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [start_rank_sequence](#)

Sequence of start ranks used in a multi-stage expansion

Specification

Alias: `rank_sequence`

Argument(s): INTEGERLIST

Default: 2

Description

Each level entry of the `start_rank_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support start rank sequences include multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [start_rank](#)

See Also

These keywords may also be of interest:

- [start_rank](#)

7.2.50.22 `adapt_rank`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [adapt_rank](#)

Activate adaptive procedure for determining best rank representation

Specification

Alias: none

Argument(s): none

Default: false

Description

The adaptive algorithm proceeds as follows:

1. Start from rank `start_rank` and form an approximation
2. Adapt the current approximation by searching for a solution with lower rank that achieves L2 accuracy within epsilon tolerance of the reference.
3. If a lower rank solution is found with comparable accuracy, then stop. If not, increase the rank by an amount specified by `kick_rank`.
4. Return to step 2 and continue until either `max_rank` is reached or a converged rank (rank less than current reference with comparable accuracy) is found.

Default Behavior

No cross validation for rank.

Examples

This example shows specification of a rank adaptation starting at rank 2, incrementing by 2, and limited at rank 10.

```
model,
  id_model = 'FT'
  surrogate global function_train
  start_order = 5
  adapt_rank start_rank = 2 kick_rank = 2 max_rank = 10
  solver_tolerance = 1e-12
  rounding_tolerance = 1e-12
  dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_rank](#)
- [kick_rank](#)
- [max_rank](#)

7.2.50.23 `kick_rank`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [kick_rank](#)

The increment in rank employed during each iteration of the rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The `adapt_rank` procedure increments the rank on each iteration, using an increment defined by `kick_rank`

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.50.24 `max_rank`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [max_rank](#)

Limits the maximum rank that is explored during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: `size_t` max

Description

The `adapt_rank` procedure increments the rank on each iteration, and the maximum value is limited by `max_rank`

Default Behavior

Max rank is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.2.50.25 max_cv_rank_candidates

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [max_cv_rank_candidates](#)

Limit the number of cross-validation candidates for rank

Specification

Alias: none

Argument(s): INTEGER

Default: size_t max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_rank](#)
- [increment_max_rank_order](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.50.26 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.50.27 sample_type

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sample Type (Group 1)	lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.50.27.1 lhs

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.50.27.2 random

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.50.28 rng

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.50.28.1 mt19937

- [Keywords Area](#)
- [method](#)

- [multilevel_function_train](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.50.28.2 rnum2

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.50.29 probability_refinement

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: `sample_refinement`

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Importance Sampling Approach (Group 1)	import	Importance sampling option for probability refinement

		adapt_import	Importance sampling option for probability refinement
		mm_adapt_import	Importance sampling option for probability refinement
	Optional	refinement_ - samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.50.29.1 import

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.50.29.2 adapt_import

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [probability_refinement](#)

- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.50.29.3 `mm_adapt_import`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.50.29.4 `refinement_samples`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the initial_samples in a sampling design.

7.2.50.30 final_moments

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.

			central	Output central moments and include them within the set of final statistics.
--	--	--	-------------------------	---

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.50.30.1 none

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [final_moments](#)

- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,  
  sampling  
    samples = 50 seed = 1234  
    response_levels = 3.6e+11 1.2e+05 3.5e+05  
    final_moments none
```

7.2.50.30.2 standard

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.50.30.3 central

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.50.31 response_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_response_levels</code>	Number of values at which to estimate desired statistics for each response
	Optional		<code>compute</code>	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num-`

`_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06

```
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06
```

Level mappings for each response function:

```
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.6000000000e+11  5.5000000000e-01
  4.0000000000e+11  3.8000000000e-01
  4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  6.0000000000e+04  6.1000000000e-01
  6.5000000000e+04  2.9000000000e-01
  7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.5000000000e+05  5.2000000000e-01
  4.0000000000e+05  9.0000000000e-02
  4.5000000000e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.50.31.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.50.31.2 compute

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.50.31.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.50.31.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.50.31.2.3 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.50.31.2.4 system

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.50.31.2.5 series

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.50.31.2.6 `parallel`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.50.32 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
```

```
no_gradients
no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.50.32.1 num_probability_levels

- [Keywords Area](#)
- [method](#)

- [multilevel_function_train](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page

7.2.50.33 reliability_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.50.33.1 num_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.50.34 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.50.34.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.50.35 `distribution`

- [Keywords Area](#)
- [method](#)

- [multilevel_function_train](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.50.35.1 cumulative

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)

- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.50.35.2 complementary

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.50.36 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

	Optional	<code>drop_tolerance</code>	Suppresses output of sensitivity indices with values lower than this tolerance
--	-----------------	-----------------------------	--

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.50.36.1 `interaction_order`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)

- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.50.36.2 `drop_tolerance`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,  
  sampling  
  sample_type lhs  
  samples = 100  
  variance_based_decomp  
  drop_tolerance = 0.001
```

7.2.50.37 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: diagonal_covariance for response vector > 10; else full_covariance

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.50.38 full_covariance

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.50.39 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.50.39.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)

- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.50.39.1.1 header

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.50.39.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.50.39.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.50.39.2 annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.50.39.3 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.50.39.4 active_only

- [Keywords Area](#)

- [method](#)
- [multilevel_function_train](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.50.40 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.50.40.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.50.40.1.1 header

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.50.40.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.50.40.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.50.40.2 annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.50.40.3 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.50.41 `seed_sequence`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For L levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 100 50 40 30 20
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.50.42 fixed_seed

- [Keywords Area](#)

- [method](#)
- [multilevel_function_train](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.50.43 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [multilevel_function_train](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.51 polynomial_chaos

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)

Uncertainty quantification using polynomial chaos expansions

Specification

Alias: nond_polynomial_chaos

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates
	Required (<i>Choose One</i>)	Chaos coefficient estimation approach (Group 1)	quadrature_order	Order for tensor-products of Gaussian quadrature rules

			sparse_grid_level	Level to use in sparse grid integration or interpolation
			cubature_integrand	Cubature using Stroud rules and their extensions
			expansion_order	The (initial) order of a polynomial expansion
			orthogonal_least_interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
			import_expansion_file	Build a Polynomial Chaos Expansion (PCE) by import coefficients and a multi-index from a file
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

	Optional	normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional	export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional	samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional	sample_type	Selection of sampling strategy
	Optional	rng	Selection of a random number generator
	Optional	probability_refinement	Allow refinement of probability and generalized reliability results using importance sampling
	Optional	final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value

	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		seed	Seed of the random number generator
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		model_pointer	Identifier for model block to be used by a method

Description

The polynomial chaos expansion (PCE) is a general framework for the approximate representation of random response functions in terms of finite-dimensional series expansions in standardized random variables

$$R = \sum_{i=0}^P \alpha_i \Psi_i(\xi)$$

where α_i is a deterministic coefficient, Ψ_i is a multidimensional orthogonal polynomial and ξ is a vector of standardized random variables. An important distinguishing feature of the methodology is that the functional relationship between random inputs and outputs is captured, not merely the output statistics as in the case of many nondeterministic methodologies.

Basis polynomial family (Group 1)

Group 1 keywords are used to select the type of basis, Ψ_i , of the expansion. Three approaches may be employed:

- Wiener: employs standard normal random variables in a transformed probability space, corresponding to Hermite orthogonal basis polynomials (see [wiener](#)).
- Askey: employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space, corresponding to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal basis polynomials, respectively (see [askey](#)).
- Extended (default if no option is selected): The Extended option avoids the use of any nonlinear variable transformations by augmenting the Askey approach with numerically-generated orthogonal polynomials for non-Askey probability density functions. Extended polynomial selections replace each of the sub-optimal Askey basis selections for bounded normal, lognormal, bounded lognormal, loguniform, triangular, gumbel, frechet, weibull, and bin-based histogram.

For supporting correlated random variables, certain fallbacks must be implemented.

- The Extended option is the default and supports only Gaussian correlations.
- If needed to support prescribed correlations (not under user control), the Extended and Askey options will fall back to the Wiener option *on a per variable basis*. If the prescribed correlations are also unsupported by Wiener expansions, then Dakota will exit with an error.

Refer to [variable_support](#) for additional information on supported variable types, with and without correlation.

Coefficient estimation approach (Group 2)

To obtain the coefficients α_i of the expansion, seven options are provided:

1. multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`).
2. multidimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`)
3. multidimensional integration by Stroud cubature rules and extensions as specified with `cubature_integrand`.
4. multidimensional integration by Latin hypercube sampling (specified with `expansion_order` and `expansion_samples`).
5. linear regression (specified with `expansion_order` and either `collocation_points` or `collocation_ratio`), using either over-determined (least squares) or under-determined (compressed sensing) approaches.
6. orthogonal least interpolation (specified with `orthogonal_least_interpolation` and `collocation_points`)
7. coefficient import from a file (specified with `import_expansion_file`). The expansion can be comprised of a general set of expansion terms, as indicated by the multi-index annotation within the file.

It is important to note that, for polynomial chaos using a single model fidelity, `quadrature_order`, `sparse_grid_level`, and `expansion_order` are scalar inputs used for a single expansion estimation. These scalars can be augmented with a `dimension_preference` to support anisotropy across the random dimension set. This differs from the use of sequence arrays in advanced use cases such as multilevel and multifidelity polynomial chaos, where multiple grid resolutions can be employed across a model hierarchy.

Active Variables

The default behavior is to form expansions over aleatory uncertain continuous variables. To form expansions over a broader set of variables, one needs to specify `active` followed by `state`, `epistemic`, `design`, or `all` in the variables specification block.

For continuous design, continuous state, and continuous epistemic uncertain variables included in the expansion, Legendre chaos bases are used to model the bounded intervals for these variables. However, these variables are not assumed to have any particular probability distribution, only that they are independent variables. Moreover, when probability integrals are evaluated, only the aleatory random variable domain is integrated, leaving behind a polynomial relationship between the statistics and the remaining design/state/epistemic variables.

Covariance type (Group 3)

These two keywords are used to specify how this method computes, stores, and outputs the covariance of the responses. In particular, the diagonal covariance option is provided for reducing post-processing overhead and output volume in high dimensional applications.

Optional Keywords regarding method outputs

Each of these sampling specifications refer to sampling on the PCE approximation for the purposes of generating approximate statistics.

- `sample_type`
- `samples`
- `seed`
- `fixed_seed`
- `rng`
- `probability_refinement`
- `distribution`
- `reliability_levels`
- `response_levels`
- `probability_levels`
- `gen_reliability_levels`

which should be distinguished from simulation sampling for generating the PCE coefficients as described in options 4, 5, and 6 above (although these options will share the `sample_type`, `seed`, and `rng` settings, if provided).

When using the `probability_refinement` control, the number of refinement samples is not under the user's control (these evaluations are approximation-based, so management of this expense is less critical). This option allows for refinement of probability and generalized reliability results using importance sampling.

Usage Tips

If n is small (e.g., two or three), then tensor-product Gaussian quadrature is quite effective and can be the preferred choice. For moderate to large n (e.g., five or more), tensor-product quadrature quickly becomes too expensive and the sparse grid and regression approaches are preferred. Random sampling for coefficient estimation is generally not recommended due to its slow convergence rate. For incremental studies, approaches 4 and 5 support reuse of previous samples through the `incremental_lhs` and `reuse_points` specifications, respectively.

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m =$

$2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension-adaptive p_refinement generalized` sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

Additional Resources

Dakota provides access to PCE methods through the `NonDPolynomialChaos` class. Refer to the Uncertainty Quantification Capabilities chapter of the Users Manual [5] and the Stochastic Expansion Methods chapter of the Theory Manual [16] for additional information on the PCE algorithm.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Integration and Expansion Moments](#)
- [Probability Density](#)
- [Level Mappings](#)
- [Variance-Based Decomposition \(Sobol' Indices\)](#)

Examples

```
method,
  polynomial_chaos
    sparse_grid_level = 2
    samples = 10000 seed = 12347 rng rnum2
    response_levels = .1 1. 50. 100. 500. 1000.
    variance_based_decomp
```

See Also

These keywords may also be of interest:

- [stoch_collocation](#)
- [function_train](#)

7.2.51.1 p_refinement

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	<code>uniform</code>	Refine an expansion uniformly in all dimensions.
			<code>dimension_adaptive</code>	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.51.1.1 uniform

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed.

7.2.51.1.2 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.51.1.2.1 sobol

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.51.1.2.2 decay

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.51.1.2.3 generalized

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)

- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,  
    polynomial_chaos  
    sparse_grid_level = 3  
    dimension_adaptive p_refinement generalized  
    max_iterations = 20  
    convergence_tol = 1.e-4
```

7.2.51.2 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
    dimension_adaptive generalized  
  p_refinement  
    max_refinement_iterations = 20  
    convergence_tol = 1.e-4  
  sparse_grid_level = 1
```

7.2.51.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)

- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.51.4 metric_scale

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	scale of refinement metric (Group 1)	relative	relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output

Usage Tips

Additional Discussion

7.2.51.4.1 relative

- [Keywords Area](#)
- [method](#)

- [polynomial_chaos](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.51.4.2 absolute

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.51.5 quadrature_order

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`, where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and [method-multifidelity_stoch_collocation-quadrature_order_sequence](#)

7.2.51.5.1 [dimension_preference](#)

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.51.5.2 [nested](#)

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.51.5.3 non_nested

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.51.6 sparse_grid_level

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.51.6.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.51.6.2 `restricted`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.51.6.3 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sparse_grid_level](#)
- `unrestricted`

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.51.6.4 `nested`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sparse_grid_level](#)

- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.51.6.5 non_nested

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.51.7 cubature_integrand

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [cubature_integrand](#)

Cubature using Stroud rules and their extensions

Specification

Alias: none

Argument(s): INTEGER

Description

Multi-dimensional integration by Stroud cubature rules [80] and extensions [94], as specified with `cubature_integrand`. A total-order expansion is used, where the isotropic order p of the expansion is half of the integrand order, rounded down. The total number of terms N for an isotropic total-order expansion of order p over n variables is given by

$$N = 1 + P = 1 + \sum_{s=1}^p \frac{1}{s!} \prod_{r=0}^{s-1} (n+r) = \frac{(n+p)!}{n!p!}$$

Since the maximum integrand order is currently five for normal and uniform and two for all other types, at most second- and first-order expansions, respectively, will be used. As a result, cubature is primarily useful for global sensitivity analysis, where the Sobol' indices will provide main effects and, at most, two-way interactions. In addition, the random variable set must be independent and identically distributed (*iid*), so the use of `askey` or `wiener` transformations may be required to create *iid* variable sets in the transformed space (as well as to allow usage of the higher order cubature rules for normal and uniform). Note that global sensitivity analysis often assumes uniform bounded regions, rather than precise probability distributions, so the *iid* restriction would not be problematic in that case.

7.2.51.8 `expansion_order`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)

The (initial) order of a polynomial expansion

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

	Required (Choose One)	Group 1		
			<code>collocation_points</code>	Number of collocation points used to estimate expansion coefficients
			<code>collocation_ratio</code>	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
			<code>expansion_samples</code>	Number of simulation samples used to estimate the expected value of a set of PCE coefficients
	Optional		<code>import_build_points_file</code>	File containing points you wish to use to build a surrogate

Description

When the `expansion_order` for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with `expansion_samples`). In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

Linear regression (specified with either `collocation_points` or `collocation_ratio`). A total-order expansion is used and must be specified using `expansion_order` as described in the previous option. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the `tensor_grid` keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If `collocation_points` or `collocation_ratio` is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined

system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the `noise_tolerance`, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders (1,...,`expansion_order`) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

7.2.51.8.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.51.8.2 `basis_type`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.51.8.2.1 [tensor_product](#)

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.51.8.2.2 total_order

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.51.8.2.3 adapted

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	advancements	The maximum number of steps used to expand a basis step.
	Optional	soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.51.8.2.4 advancements

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.51.8.2.5 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)

- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.51.8.3 collocation_points

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares

		orthogonal_-matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit_-denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle_-regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_-shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation_ratio-collocation_points_sequence

7.2.51.8.3.1 least_squares

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group LSQ Regression Approach (Group 1)	Dakota Keyword svd	Dakota Keyword Description Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_ constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.51.8.3.2 svd

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.51.8.3.3 equality_constrained

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.51.8.3.4 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.51.8.3.5 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.3.6 basis_pursuit

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.51.8.3.7 basis_pursuit_denoising

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.51.8.3.8 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.3.9 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.51.8.3.10 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)

- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.3.11 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.51.8.3.12 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.3.13 l2_penalty

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.51.8.3.14 cross_validation

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.51.8.3.15 noise_only

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.51.8.3.16 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)

- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.51.8.3.17 ratio_order

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces $\text{samples} = 2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.51.8.3.18 response_scaling

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.51.8.3.19 use_derivatives

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.51.8.3.20 tensor_grid

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)

- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.51.8.3.21 reuse_points

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_points](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.51.8.3.22 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)

- [collocation_points](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.51.8.4 [collocation_ratio](#)

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit - denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle - regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute - shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.51.8.4.1 least_squares

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group LSQ Regression Approach (Group 1)	Dakota Keyword svd	Dakota Keyword Description Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_ constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.51.8.4.2 svd

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.51.8.4.3 equality_constrained

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.51.8.4.4 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.51.8.4.5 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.4.6 basis_pursuit

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.51.8.4.7 basis_pursuit_denoising

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.51.8.4.8 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.4.9 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.51.8.4.10 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.4.11 `least_absolute_shrinkage`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.51.8.4.12 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.51.8.4.13 l2_penalty

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.51.8.4.14 cross_validation

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.51.8.4.15 noise_only

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.51.8.4.16 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)

- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.51.8.4.17 ratio_order

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces $\text{samples} = 2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.51.8.4.18 response_scaling

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.51.8.4.19 use_derivatives

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.51.8.4.20 tensor_grid

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)

- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.51.8.4.21 reuse_points

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.51.8.4.22 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)

- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.51.8.5 [expansion_samples](#)

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [expansion_samples](#)

Number of simulation samples used to estimate the expected value of a set of PCE coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Description

In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

A corresponding sequence specification is documented at, e.g., [expansion_samples_sequence](#) and [expansion_samples_sequence](#)

See Also

These keywords may also be of interest:

- [expansion_samples_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-expansion_samples_sequence

7.2.51.8.5.1 reuse_points

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [expansion_samples](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.51.8.6 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.51.8.6.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:


```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1      x2      obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9      1.1      0.0002      0.26      0.76
2             0.90009  1.1 0.0001996404857  0.2601620081  0.759955
3             0.89991  1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.51.8.6.2 header

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.51.8.6.3 eval_id

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.51.8.6.4 interface_id

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.51.8.6.5 annotated

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.8.6.6 freeform

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.8.6.7 active_only

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.51.9 orthogonal_least_interpolation

- [Keywords Area](#)
- [method](#)

- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: least_interpolation oli

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points	Number of collocation points used to estimate expansion coefficients
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.51.9.1 `collocation_points`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation_ratio-collocation_points_sequence

7.2.51.9.2 tensor_grid

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.51.9.3 reuse_points

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.51.9.4 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g.

surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.51.9.4.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	header	Enable header row in custom-annotated tabular file
	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1         0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.51.9.4.2 header

- [Keywords Area](#)
- [method](#)

- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.51.9.4.3 `eval_id`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.51.9.4.4 interface_id

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.51.9.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.9.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009      1.1 0.0001996404857  0.2601620081  0.759955
0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.51.9.4.7 active_only

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.51.10 `import_expansion_file`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [import_expansion_file](#)

Build a Polynomial Chaos Expansion (PCE) by import coefficients and a multi-index from a file

Specification

Alias: none

Argument(s): STRING

Description

The coefficients can be specified in an arbitrary order. The multi-index provided is used to generate a sparse expansion that consists only of the indices corresponding to the non-zero coefficients provided in the file.

7.2.51.11 `askey`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.51.12 wiener

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.51.13 normalized

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.51.14 export_expansion_file

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.51.15 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.51.16 sample_type

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sample Type (Group 1)	lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.51.16.1 lhs

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.51.16.2 random

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.51.17 rng

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.51.17.1 mt19937

- [Keywords Area](#)
- [method](#)

- [polynomial_chaos](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.51.17.2 rnum2

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.51.18 probability_refinement

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: `sample_refinement`

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Importance Sampling Approach (Group 1)	import	Importance sampling option for probability refinement

		adapt_import	Importance sampling option for probability refinement
		mm_adapt_import	Importance sampling option for probability refinement
	Optional	refinement_ - samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.51.18.1 import

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.51.18.2 adapt_import

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [probability_refinement](#)

- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.51.18.3 `mm_adapt_import`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.51.18.4 `refinement_samples`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the initial_samples in a sampling design.

7.2.51.19 final_moments

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.

			central	Output central moments and include them within the set of final statistics.
--	--	--	-------------------------	---

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.51.19.1 none

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [final_moments](#)

- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,  
  sampling  
    samples = 50 seed = 1234  
    response_levels = 3.6e+11 1.2e+05 3.5e+05  
    final_moments none
```

7.2.51.19.2 standard

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.51.19.3 central

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.51.20 response_levels

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_response_levels</code>	Number of values at which to estimate desired statistics for each response
	Optional		<code>compute</code>	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num-`

`_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06

```
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06
```

Level mappings for each response function:

```
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.6000000000e+11  5.5000000000e-01
  4.0000000000e+11  3.8000000000e-01
  4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  6.0000000000e+04  6.1000000000e-01
  6.5000000000e+04  2.9000000000e-01
  7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.5000000000e+05  5.2000000000e-01
  4.0000000000e+05  9.0000000000e-02
  4.5000000000e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.51.20.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.51.20.2 compute

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.51.20.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.51.20.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.51.20.2.3 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.51.20.2.4 system

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.51.20.2.5 series

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.51.20.2.6 `parallel`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.51.21 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds    = 199.3, 474.63
  upper_bounds    = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs       = 3 4
  abscissas       = 5 8 10 .1 .2 .3 .4
  counts          = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs       = 2
  abscissas       = 3 4
  counts          = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
```

```
no_gradients
no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.51.21.1 num_probability_levels

- [Keywords Area](#)
- [method](#)

- [polynomial_chaos](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page

7.2.51.22 reliability_levels

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.51.22.1 num_reliability_levels

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.51.23 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.51.23.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.51.24 `distribution`

- [Keywords Area](#)
- [method](#)

- [polynomial_chaos](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.51.24.1 cumulative

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)

- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.51.24.2 complementary

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.51.25 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

	Optional	<code>drop_tolerance</code>	Suppresses output of sensitivity indices with values lower than this tolerance
--	-----------------	-----------------------------	--

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.51.25.1 `interaction_order`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)

- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.51.25.2 `drop_tolerance`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,  
  sampling  
  sample_type lhs  
  samples = 100  
  variance_based_decomp  
  drop_tolerance = 0.001
```

7.2.51.26 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: diagonal_covariance for response vector > 10; else full_covariance

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.51.27 full_covariance

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.51.28 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.51.28.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)

- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.28.1.1 header

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.51.28.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.51.28.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.51.28.2 annotated

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.28.3 freeform

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.28.4 active_only

- [Keywords Area](#)

- [method](#)
- [polynomial_chaos](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.51.29 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.51.29.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.51.29.1.1 header

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.51.29.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.51.29.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.51.29.2 annotated

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.29.3 freeform

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.51.30 seed

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.51.31 `fixed_seed`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The

same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.51.32 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [polynomial_chaos](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                      0.1 0.2 0.6
                      0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.52 multifidelity_polynomial_chaos

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)

Multifidelity uncertainty quantification using polynomial chaos expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates
	Optional		statistics_mode	type of statistical metric roll-up for multifidelity UQ methods
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_ - emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)	Chaos Coefficient Estimation Approach (Group 1)	quadrature_order_ - sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_ - sequence	Sequence of sparse grid levels used in a multi-stage expansion

			expansion_order_sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

	Optional	samples_on_ - emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional	sample_type	Selection of sampling strategy
	Optional	rng	Selection of a random number generator
	Optional	probability_ - refinement	Allow refinement of probability and generalized reliability results using importance sampling
	Optional	final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional	gen_reliability_ - levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions

	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		model_pointer	Identifier for model block to be used by a method

Description

As described in [polynomial_chaos](#), the polynomial chaos expansion (PCE) is a general framework for the approximate representation of random response functions in terms of series expansions in standardized random variables:

$$R = \sum_{i=0}^P \alpha_i \Psi_i(\xi)$$

where α_i is a deterministic coefficient, Ψ_i is a multidimensional orthogonal polynomial and ξ is a vector of standardized random variables.

In the multilevel and multifidelity cases, we decompose this expansion into several constituent expansions, one per model form or solution control level. In a bi-fidelity case with low-fidelity (LF) and high-fidelity (HF) models and an additive discrepancy approach, we have:

$$R = \sum_{i=0}^{p^{LF}} \alpha_i^{LF} \Psi_i(\xi) + \sum_{i=0}^{p^{HF}} \delta_i \Psi_i(\xi)$$

where δ_i is a coefficient for the discrepancy expansion.

The same specification options are available as described in [polynomial_chaos](#) with one key difference: many of the coefficient estimation inputs change from a scalar input for a single expansion to a *sequence* specification for a low-fidelity expansion followed by multiple discrepancy expansions.

To obtain the coefficients α_i and δ_i for each of the expansions, the following options are provided:

1. multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order_sequence`, and, optionally, `dimension_preference`).
2. multidimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level_sequence` and, optionally, `dimension_preference`)
3. multidimensional integration by Latin hypercube sampling (specified with `expansion_order_sequence` and `expansion_samples_sequence`).
4. linear regression (specified with `expansion_order_sequence` and either `collocation_points_sequence` or `collocation_ratio`), using either over-determined (least squares) or under-determined (compressed sensing) approaches.
5. orthogonal least interpolation (specified with `orthogonal_least_interpolation` and `collocation_points_sequence`)

It is important to note that, while `quadrature_order_sequence`, `sparse_grid_level_sequence`, `expansion_order_sequence`, `expansion_samples_sequence`, and `collocation_points_sequence` are array inputs, only one scalar from these arrays is active at a time for a particular expansion estimation. In order to specify anisotropy in resolution across the random variable set, a `dimension_preference` specification can be used to augment scalar specifications for quadrature order, sparse grid level, and expansion order.

Multifidelity UQ using PCE requires that the model selected for iteration by the method specification is a multifidelity surrogate model (see [hierarchical](#)), which defines an `ordered_model_sequence` (see [hierarchical](#)). Two types of hierarchies are supported: (i) a hierarchy of model forms composed from more than one model within the `ordered_model_sequence`, or (ii) a hierarchy of discretization levels comprised from a single model within the `ordered_model_sequence` which in turn specifies a `solution_level_control` (see [solution_level_control](#)).

In both cases, an expansion will first be formed for the low fidelity model or coarse discretization, using the first value within the coefficient estimation sequence, along with any specified refinement strategy. Second, expansions are formed for one or more model discrepancies (the difference between response results if `additive_correction` or the ratio of results if `multiplicative_correction`), using all subsequent values in the coefficient estimation sequence (if the sequence does not provide a new value, then the previous value is reused) along with any specified refinement strategy. The number of discrepancy expansions is determined by the number of model forms or discretization levels in the hierarchy.

After formation and refinement of the constituent expansions, each of the expansions is combined (added or multiplied) into an expansion that approximates the high fidelity model, from which the final set of statistics are generated. For polynomial chaos expansions, this high fidelity expansion can differ significantly in form from the low fidelity and discrepancy expansions, particularly in the `multiplicative` case where it is expanded to include all of the basis products.

Additional Resources

Dakota provides access to multifidelity PCE methods through the `NonDMultilevelPolynomialChaos` class. Refer to the Stochastic Expansion Methods chapter of the Theory Manual [16] for additional information on the Multifidelity PCE algorithm.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Integration and Expansion Moments](#) (expansion moments only)

- [Probability Density](#)
- [Level Mappings](#)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

Examples

```
method,
  multifidelity_polynomial_chaos
  model_pointer = 'HIERARCH'
  sparse_grid_level_sequence = 4 3 2

model,
  id_model = 'HIERARCH'
  surrogate hierarchical
  ordered_model_fidelities = 'LF' 'MF' 'HF'
  correction additive zeroth_order
```

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [multifidelity_stoch_collocation](#)
- [multifidelity_function_train](#)

7.2.52.1 p_refinement

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.

			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
--	--	--	------------------------------------	--

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.52.1.1 uniform

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.52.1.2 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.52.1.2.1 sobol

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.52.1.2.2 decay

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.52.1.2.3 generalized

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.52.2 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.52.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.52.4 metric_scale

- [Keywords Area](#)
- [method](#)

- [multifidelity_polynomial_chaos](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group scale of refinement metric (Group 1)	Dakota Keyword relative	Dakota Keyword Description relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output

Usage Tips

Additional Discussion

7.2.52.4.1 `relative`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.52.4.2 absolute

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.52.5 statistics_mode

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [statistics_mode](#)

type of statistical metric roll-up for multifidelity UQ methods

Specification

Alias: none

Argument(s): none

Default: DEFAULT_EXPANSION_STATS

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics mode for refinement metric (Group 1)	Dakota Keyword active	Dakota Keyword Description use statistics for the active expansion within multifidelity refinement
			combined	use statistics for the combined expansion within multifidelity refinement

Description

When performing adaptive refinement in multifidelity UQ methods, one can adapt the expansion for a model level/fidelity based on its local statistics ([active](#)) or based on the statistics for the aggregate expansion ([combined](#)). The latter case may incur additional computational overhead, but ensures that the refinement occurs in the aggregate multifidelity context.

Default Behavior

Default for multifidelity methods is [combined](#) statistics.

Expected Output

Usage Tips

Additional Discussion

7.2.52.5.1 [active](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [statistics_mode](#)
- [active](#)

use statistics for the active expansion within multifidelity refinement

Specification

Alias: none

Argument(s): none

Description

When performing adaptive refinement in multifidelity UQ methods, this option adapts based on the statistics for the active model level/fidelity. This approach avoids some computational overhead in some cases by avoiding expansion roll-up.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.52.5.2 combined

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [statistics_mode](#)
- [combined](#)

use statistics for the combined expansion within multifidelity refinement

Specification

Alias: none

Argument(s): none

Description

When performing adaptive refinement in multifidelity UQ methods, this option adapts based on the statistics for the aggregate multifidelity expansion. This approach may incur additional computational overhead, but ensures that the refinement occurs in the aggregate multifidelity context.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.52.6 allocation_control

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity polynomial chaos

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.52.6.1 [greedy](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multifidelity polynomial chaos supports greedy refinement strategies, spanning regression and projection approaches for computing expansion coefficients. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within a unified competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any $z/p/\beta/\beta^*$ level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity regression starts from a zeroth-order reference expansion (a constant) for each level, and generates candidate refinements for each level that are competed in an integrated greedy competition. The number of new samples for the incremented candidate expansion order is determined from the collocation ratio. In this case, the number of candidates for each level is limited to one uniform refinement of the current expansion order.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement uniform
  expansion_order_sequence = 0
  collocation_ratio = .9 seed = 160415
  orthogonal_matching_pursuit
  convergence_tolerance 1.e-2
```

The next example employs generalized sparse grids within a greedy multifidelity competition. Each modeling level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all model levels is competed within an integrated greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each model level and the number of candidates grows rapidly with random dimension and grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement dimension_adaptive generalized
  sparse_grid_level_sequence = 0 unrestricted
  convergence_tolerance 1.e-8
```

7.2.52.7 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Discrepancy Emulation Approach (Group 1)	distinct	Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
    multilevel_polynomial_chaos
    expansion_order_sequence = 2
    collocation_ratio = .9
    orthogonal_matching_pursuit
    discrepancy_emulation recursive
```

7.2.52.7.1 `distinct`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.52.7.2 recursive

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.52.8 quadrature_order_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.52.8.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.52.8.2 nested

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.52.8.3 non_nested

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.52.9 `sparse_grid_level_sequence`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.52.9.1 dimension_preference

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.52.9.2 restricted

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sparse_grid_level_sequence](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.52.9.3 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sparse_grid_level_sequence](#)
- `unrestricted`

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.52.9.4 `nested`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sparse_grid_level_sequence](#)

- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.52.9.5 non_nested

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.52.10 expansion_order_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification**Alias:** none**Argument(s):** INTEGERLIST**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	<p>Optional</p> <p>Optional</p> <p>Required (<i>Choose One</i>)</p>	<p>Group 1</p>	<p>dimension_ - preference</p> <p>basis_type</p> <p>collocation_ratio</p>	<p>A set of weights specifying the relative importance of each uncertain variable (dimension)</p> <p>Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.</p> <p>Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.</p>
			<p>expansion_ - samples_sequence</p>	<p>Sequence of expansion samples used in a multi-stage polynomial chaos expansion</p> <p>Each level entry of the <code>expansion_ - samples_ - sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.</p> <p>A corresponding scalar specification is documented at, e.g., expansion_ - samples</p>
<p>Dakota Version 6.16 Reference Manual updated on May 12, 2022</p>	<p>Optional</p>		<p>import_build_ - points_file</p>	<p>File containing points you wish to use to build a surrogate</p>

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.52.10.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.52.10.2 `basis_type`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)

- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.52.10.2.1 [tensor_product](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.52.10.2.2 total_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.52.10.2.3 adapted

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence-limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.52.10.2.4 advancements

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.52.10.2.5 `soft_convergence_limit`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.52.10.3 `collocation_ratio`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (Choose One)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_ matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
			basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
			basis_pursuit_ denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
			least_angle_ regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
			least_absolute_ shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.

	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_ - iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.52.10.3.1 collocation_points_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.52.10.3.2 least_squares

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.52.10.3.3 [svd](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.52.10.3.4 [equality_constrained](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.52.10.3.5 [orthogonal_matching_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.52.10.3.6 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.52.10.3.7 basis_pursuit

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.52.10.3.8 basis_pursuit_denoising

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.52.10.3.9 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.52.10.3.10 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.52.10.3.11 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.52.10.3.12 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.52.10.3.13 noise_tolerance

- [Keywords Area](#)
- [method](#)

- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.52.10.3.14 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.52.10.3.15 cross_validation

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.52.10.3.16 noise_only

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.52.10.3.17 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)

- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.52.10.3.18 ratio_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.52.10.3.19 response_scaling

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.52.10.3.20 `use_derivatives`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.52.10.3.21 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.52.10.3.22 reuse_points

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.52.10.3.23 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)

- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.52.10.4 [expansion_samples_sequence](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.52.10.4.1 reuse_points

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.52.10.5 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.52.10.5.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.52.10.5.2 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.52.10.5.3 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.52.10.5.4 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.52.10.5.5 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9         1.1         0.0002          0.26          0.76
2          NO_ID           0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```


7.2.52.10.5.6 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.52.10.5.7 active_only

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.52.11 orthogonal_least_interpolation

- [Keywords Area](#)
- [method](#)

- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: least_interpolation oli

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.52.11.1 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.52.11.2 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.52.11.3 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)

- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.52.11.4 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.52.11.4.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009        1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991        1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.52.11.4.2 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.52.11.4.3 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.52.11.4.4 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.52.11.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.52.11.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.52.11.4.7 active_only

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.52.12 `askey`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.52.13 wiener

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.52.14 normalized

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.52.15 export_expansion_file

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.52.16 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
  quadrature_order = 2
  samples_on_emulator = 10000
```

7.2.52.17 sample_type

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Sample Type (Group 1)	Dakota Keyword lhs	Dakota Keyword Description Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.52.17.1 lhs

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.52.17.2 random

- [Keywords Area](#)
- [method](#)

- [multifidelity_polynomial_chaos](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.52.18 rng

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.52.18.1 `mt19937`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.52.18.2 rnum2

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.52.19 `probability_refinement`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: `sample_refinement`

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group Importance Sampling Approach (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.52.19.1 import

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.52.19.2 adapt_import

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [probability_refinement](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.52.19.3 mm_adapt_import

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.52.19.4 refinement_samples

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.52.20 `final_moments`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
```

```

samples = 50 seed = 1234
response_levels = 3.6e+11 1.2e+05 3.5e+05
final_moments central

```

Overriding the default to `none` as follows:

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```

allows associated nested model mappings to be simplified from:

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.

```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.

```

7.2.52.20.1 none

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```


7.2.52.20.2 standard

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.52.20.3 central

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.52.21 response_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds    = 199.3, 474.63
  upper_bounds    = 298.5, 712.
  descriptors     = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors     = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs      = 3 4
  abscissas      = 5 8 10 .1 .2 .3 .4
  counts         = 17 21 0 12 24 12 0
  descriptors    = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs    = 2
  abscissas    = 3 4
  counts       = 1 1
  descriptors  = 'TF3h'
```

```

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06
4.0000000000e+05	4.5000000000e+05	1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
3.6000000000e+11	5.5000000000e-01		
4.0000000000e+11	3.8000000000e-01		
4.4000000000e+11	2.3000000000e-01		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
6.0000000000e+04	6.1000000000e-01		
6.5000000000e+04	2.9000000000e-01		
7.0000000000e+04	9.0000000000e-02		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
3.5000000000e+05	5.2000000000e-01		
4.0000000000e+05	9.0000000000e-02		
4.5000000000e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.52.21.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.52.21.2 compute

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
```

```

                                3.5e+05 4.0e+05 4.5e+05
compute reliabilities

```

7.2.52.21.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```

method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities

```

7.2.52.21.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)

- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.52.21.2.3 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.52.21.2.4 system

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system

			parallel	Aggregate response statistics assuming a parallel system
--	--	--	--------------------------	--

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.52.21.2.5 series

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.52.21.2.6 parallel

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.52.22 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```

method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors       = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

```

PDF for response_fn_1:
  Bin Lower          Bin Upper          Density Value
  -----
  2.7604749078e+11  3.4221494996e+11  5.1384774972e-12
  3.4221494996e+11  4.0634975300e+11  5.1454122311e-12
  4.0634975300e+11  5.4196114379e+11  2.4334239039e-12

```

```

PDF for response_fn_2:
  Bin Lower          Bin Upper          Density Value
  -----
  4.6431154744e+04  5.6511827775e+04  1.9839945149e-05
  5.6511827775e+04  6.1603813790e+04  5.8916108390e-05
  6.1603813790e+04  7.8702465755e+04  2.9242071306e-05

```

```

PDF for response_fn_3:
  Bin Lower          Bin Upper          Density Value
  -----
  2.3796737090e+05  3.6997214153e+05  5.3028386523e-06
  3.6997214153e+05  3.8100966235e+05  9.0600055634e-06
  3.8100966235e+05  4.4111498127e+05  3.3274925348e-06

```

Level mappings for each response function:

```

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  2.7604749078e+11  1.0000000000e+00
  3.4221494996e+11  6.6000000000e-01

```

```

4.0634975300e+11  3.3000000000e-01
5.4196114379e+11  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
4.6431154744e+04  1.0000000000e+00
5.6511827775e+04  8.0000000000e-01
6.1603813790e+04  5.0000000000e-01
7.8702465755e+04  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
2.3796737090e+05  1.0000000000e+00
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.4111498127e+05  0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.52.22.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.52.23 reliability_levels

- [Keywords Area](#)
- [method](#)

- [multifidelity_polynomial_chaos](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.52.23.1 `num_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: reliability_levels evenly distributed among response functions

Description

See parent page

7.2.52.24 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.52.24.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.52.25 distribution

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Computes statistics according to cumulative functions

			complementary	Computes statistics according to complementary cumulative functions
--	--	--	-------------------------------	---

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.52.25.1 cumulative

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.52.25.2 complementary

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.52.26 variance_based_decomp

- [Keywords Area](#)

- [method](#)
- [multifidelity_polynomial_chaos](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.52.26.1 interaction_order

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.52.26.2 drop_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
    sample_type lhs
    samples = 100
    variance_based_decomp
    drop_tolerance = 0.001
```

7.2.52.27 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.52.28 full_covariance

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.52.29 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.52.29.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
--	-------------------	----------------------	----------------	----------------------------

	Optional	header	Enable header row in custom-annotated tabular file
	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1         0.0002  0.26            0.76
2              0.90009    1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991    1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.52.29.1.1 header

- [Keywords Area](#)
- [method](#)

- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.52.29.1.2 `eval_id`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.52.29.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)

- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.52.29.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.52.29.3 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.52.29.4 active_only

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.52.30 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.52.30.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: `annotated` format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.52.30.1.1 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.52.30.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.52.30.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.52.30.2 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.52.30.3 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

          0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...

```

7.2.52.31 seed_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For L levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 100 50 40 30 20
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.52.32 `fixed_seed`

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.52.33 model_pointer

- [Keywords Area](#)
- [method](#)
- [multifidelity_polynomial_chaos](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```

environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.53 multilevel_polynomial_chaos

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)

Multilevel uncertainty quantification using polynomial chaos expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		allocation_control	Sample allocation approach for multilevel expansions
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates
	Optional		discrepancy_emulation	Formulation for emulation of model discrepancies.
	Required (Choose One)	Coefficient Computation Approach (Group 1)	expansion_order_sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (Choose One)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

	Optional	normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional	export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional	samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional	sample_type	Selection of sampling strategy
	Optional	rng	Selection of a random number generator
	Optional	probability_refinement	Allow refinement of probability and generalized reliability results using importance sampling
	Optional	final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value

	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets

	Optional	<code>model_pointer</code>	Identifier for model block to be used by a method
--	-----------------	----------------------------	---

Description

As described in [polynomial_chaos](#), the polynomial chaos expansion (PCE) is a general framework for the approximate representation of random response functions in terms of series expansions in standardized random variables:

$$R = \sum_{i=0}^P \alpha_i \Psi_i(\xi)$$

where α_i is a deterministic coefficient, Ψ_i is a multidimensional orthogonal polynomial and ξ is a vector of standardized random variables.

In the multilevel and multifidelity cases, we decompose this expansion into several constituent expansions, one per model form or solution control level. In a bi-fidelity case with low-fidelity (LF) and high-fidelity (HF) models, we have:

$$R = \sum_{i=0}^{p^{LF}} \alpha_i^{LF} \Psi_i(\xi) + \sum_{i=0}^{p^{HF}} \delta_i \Psi_i(\xi)$$

where δ_i is a coefficient for the discrepancy expansion.

For the case of regression-based PCE (least squares, compressed sensing, or orthogonal least interpolation), an optimal sample allocation procedure can be applied for the resolution of each level within a multilevel sampling procedure as in [multilevel_sampling](#). The core difference is that a Monte Carlo estimator of the statistics is replaced with a PCE-based estimator of the statistics, requiring approximation of the variance of these estimators.

Initial prototypes for multilevel PCE can be explored using `dakota/share/dakota/test/dakota_uq_diffusion_mlpce.in`, and will be stabilized in future releases.

Additional Resources

Dakota provides access to multilevel PCE methods through the `NonDMultilevelPolynomialChaos` class. Refer to the Stochastic Expansion Methods chapter of the Theory Manual [16] for additional information on the Multilevel PCE algorithm.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Integration and Expansion Moments](#) (expansion moments only)
- [Probability Density](#)
- [Level Mappings](#)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

Examples

```
method,
  multilevel_polynomial_chaos
  model_pointer = 'HIERARCH'
  pilot_samples = 10
  expansion_order_sequence = 2
  collocation_ratio = .9
  seed = 1237
```

```
    orthogonal_matching_pursuit
    convergence_tolerance = .01
    output silent

model,
  id_model = 'HIERARCH'
  surrogate hierarchical
  ordered_model_fidelities = 'SIM1'
  correction additive zeroth_order

model,
  id_model = 'SIM1'
  simulation
  solution_level_control = 'mesh_size'
  solution_level_cost = 1. 8. 64. 512. 4096.
```

See Also

These keywords may also be of interest:

- [adaptive_sampling](#)
- [gpais](#)
- [local_reliability](#)
- [global_reliability](#)
- [sampling](#)
- [importance_sampling](#)
- [stoch_collocation](#)

7.2.53.1 max_iterations

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.53.2 allocation_control

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [allocation_control](#)

Sample allocation approach for multilevel expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multilevel Sample Allocation Control (Group 1)	estimator_variance	Variance of mean estimator within multilevel polynomial chaos
			rip_sampling	Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Description

Multilevel expansions, including regression-based polynomial chaos expansion (PCE) and function train (FT) expansions, require a sample allocation strategy. Three options are currently available:

- allocation based on assuming a convergence rate for the estimator variance (for regression PCE)

- restricted isometry property (RIP) sampling (for regression PCE via compressed sensing)
- rank sampling (for FT)

Default Behavior

Current defaults, when `allocation_control` is not specified, are estimator variance for PCE and rank sampling for FT.

7.2.53.2.1 estimator_variance

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [allocation_control](#)
- [estimator_variance](#)

Variance of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Optional		estimator_rate	Rate of convergence of mean estimator within multilevel polynomial chaos

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ , with default values that may be overridden as part of this specification block.

This approach is supported for regression-based PCE approaches (over-determined least squares, under-determined compressed sensing, or orthogonal least interpolation).

In practice, it can be challenging to estimate a smooth convergence rate for estimator variance in the presence of abrupt transitions in the quality of sparse recoveries. As a result, sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the convergence of the estimator variance, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```
method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
    orthogonal_matching_pursuit
    expansion_order_sequence = 2
    pilot_samples = 10
    collocation_ratio = .9
    allocation_control
      estimator_variance estimator_rate = 2.5
    seed = 1237
    convergence_tolerance = .01
```

7.2.53.2.1.1 estimator_rate

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [allocation_control](#)
- [estimator_variance](#)
- [estimator_rate](#)

Rate of convergence of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): REAL

Default: 2

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ .

The default values are $\gamma = 1$ and $\kappa = 2$ (adopts a more aggressive sample profile by assuming a faster convergence rate than Monte Carlo). This advanced specification option allows to user to specify κ , overriding the default.

7.2.53.2.2 rip_sampling

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [allocation_control](#)
- [rip_sampling](#)

Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multilevel polynomial chaos with compressed sensing may allocate the number of samples per level based on the restricted isometry property (RIP), applied to recovery at level l :

$$N_l \geq s_l \log^3(s_l) L_l \log(C_l)$$

for sparsity s , cardinality C , and mutual coherence L . The adaptive algorithm starts from a pilot sample, shapes the profile based on observed sparsity, and iterates until convergence. In practice, RIP sampling levels are quite conservative, and a collocation ratio constraint ($N_l \leq rC_l$, where r defaults to 2) must be enforced on the profile.

The algorithm relies on observed sparsity, it is appropriate for use with regularized solvers for compressed sensing. It employs orthogonal matching pursuit (OMP) by default and automatically activates cross-validation in order to choose the best noise parameter value for the recovery.

This capability is **experimental**. Sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the number of sparse coefficient sets recovered for each level, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```
method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
  orthogonal_matching_pursuit
  expansion_order_sequence = 2
  pilot_samples = 10
  collocation_ratio = .9
  allocation_control rip_sampling
  seed = 1237
  convergence_tolerance = .01
```

7.2.53.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)

- [multilevel_polynomial_chaos](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.53.4 `metric_scale`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group scale of refinement metric (Group 1)	Dakota Keyword relative	Dakota Keyword Description relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output

Usage Tips

Additional Discussion

7.2.53.4.1 relative

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.53.4.2 `absolute`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.53.5 `discrepancy_emulation`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword distinct	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.53.5.1 distinct

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [discrepancy_emulation](#)

- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.53.5.2 recursive

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.53.6 expansion_order_sequence

- [Keywords Area](#)
- [method](#)

- [multilevel_polynomial_chaos](#)

- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional	Group 1	dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.
	Required (Choose One)		collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
			expansion_samples_sequence	Sequence of expansion samples used in a multi-stage polynomial chaos expansion. Each level entry of the <code>expansion_samples_sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach. A corresponding scalar specification is documented at, e.g., expansion_samples
	Optional		points_file	File containing points you wish to use to build a surrogate

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.53.6.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.53.6.2 `basis_type`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)

- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.53.6.2.1 [tensor_product](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.53.6.2.2 total_order

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.53.6.2.3 adapted

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.53.6.2.4 advancements

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.53.6.2.5 `soft_convergence_limit`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.53.6.3 `collocation_ratio`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (Choose One)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_ matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
			basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
			basis_pursuit_ denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
			least_angle_ regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
			least_absolute_ shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.

	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_ iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.53.6.3.1 collocation_points_sequence

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.53.6.3.2 least_squares

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.53.6.3.3 [svd](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.53.6.3.4 [equality_constrained](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.53.6.3.5 [orthogonal_matching_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.53.6.3.6 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.53.6.3.7 basis_pursuit

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.53.6.3.8 `basis_pursuit_denoising`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: `bpdn`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.53.6.3.9 `noise_tolerance`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.53.6.3.10 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.53.6.3.11 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.53.6.3.12 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.53.6.3.13 noise_tolerance

- [Keywords Area](#)
- [method](#)

- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.53.6.3.14 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.53.6.3.15 cross_validation

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.53.6.3.16 noise_only

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.53.6.3.17 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)

- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.53.6.3.18 ratio_order

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces $\text{samples} = 2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.53.6.3.19 response_scaling

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.53.6.3.20 use_derivatives

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.53.6.3.21 tensor_grid

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.53.6.3.22 reuse_points

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.53.6.3.23 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)

- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.53.6.4 [expansion_samples_sequence](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.53.6.4.1 reuse_points

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.53.6.5 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.53.6.5.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1      x2      obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9      1.1      0.0002      0.26      0.76
2             0.90009  1.1 0.0001996404857  0.2601620081  0.759955
3             0.89991  1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.53.6.5.2 header

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.53.6.5.3 eval_id

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.53.6.5.4 interface_id

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.53.6.5.5 annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.53.6.5.6 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.53.6.5.7 active_only

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.53.7 orthogonal_least_interpolation

- [Keywords Area](#)
- [method](#)

- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: least_interpolation oli

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.53.7.1 collocation_points_sequence

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.53.7.2 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.53.7.3 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)

- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.53.7.4 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.53.7.4.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```


7.2.53.7.4.2 header

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.53.7.4.3 eval_id

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.53.7.4.4 interface_id

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.53.7.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.53.7.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009      1.1 0.0001996404857  0.2601620081  0.759955
0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.53.7.4.7 active_only

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.53.8 askey

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.53.9 wiener

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.53.10 normalized

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.53.11 export_expansion_file

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.53.12 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.53.13 sample_type

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Sample Type (Group 1)	Dakota Keyword lhs	Dakota Keyword Description Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.53.13.1 lhs

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.53.13.2 random

- [Keywords Area](#)
- [method](#)

- [multilevel_polynomial_chaos](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.53.14 rng

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.53.14.1 mt19937

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.53.14.2 `rnum2`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.53.15 probability_refinement

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: sample_refinement

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group Importance Sampling Approach (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.53.15.1 import

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.53.15.2 adapt_import

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [probability_refinement](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.53.15.3 mm_adapt_import

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.53.15.4 refinement_samples

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.53.16 `final_moments`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
```



```

samples = 50 seed = 1234
response_levels = 3.6e+11 1.2e+05 3.5e+05
final_moments central

```

Overriding the default to `none` as follows:

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```

allows associated nested model mappings to be simplified from:

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.

```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.

```

7.2.53.16.1 none

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```

7.2.53.16.2 standard

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.53.16.3 central

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.53.17 response_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'
```

```

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06
4.0000000000e+05	4.5000000000e+05	1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
3.6000000000e+11	5.5000000000e-01		
4.0000000000e+11	3.8000000000e-01		
4.4000000000e+11	2.3000000000e-01		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
6.0000000000e+04	6.1000000000e-01		
6.5000000000e+04	2.9000000000e-01		
7.0000000000e+04	9.0000000000e-02		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
3.5000000000e+05	5.2000000000e-01		
4.0000000000e+05	9.0000000000e-02		
4.5000000000e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.53.17.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.53.17.2 compute

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
```

```

                                3.5e+05 4.0e+05 4.5e+05
compute reliabilities

```

7.2.53.17.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```

method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities

```

7.2.53.17.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)

- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.53.17.2.3 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.53.17.2.4 system

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.53.17.2.5 series

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.53.17.2.6 parallel

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.53.18 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.
```

```

variables,
  normal_uncertain = 2
    means          = 248.89, 593.33
    std_deviations = 12.4, 29.7
    descriptors    = 'TF1n' 'TF2n'
  uniform_uncertain = 2
    lower_bounds   = 199.3, 474.63
    upper_bounds   = 298.5, 712.
    descriptors    = 'TF1u' 'TF2u'
  weibull_uncertain = 2
    alphas         = 12., 30.
    betas          = 250., 590.
    descriptors    = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
    num_pairs      = 3 4
    abscissas     = 5 8 10 .1 .2 .3 .4
    counts        = 17 21 0 12 24 12 0
    descriptors    = 'TF1h' 'TF2h'
  histogram_point_uncertain
    real = 1
    num_pairs     = 2
    abscissas     = 3 4
    counts        = 1 1
    descriptors    = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		

```

6.1603813790e+04  5.0000000000e-01
7.8702465755e+04  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
2.3796737090e+05  1.0000000000e+00
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.4111498127e+05  0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.53.18.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.53.19 reliability_levels

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.53.19.1 `num_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.53.20 `gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_gen_reliability_levels</code>	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.53.20.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability_levels evenly distributed among response functions

Description

See parent page

7.2.53.21 distribution

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.53.21.1 cumulative

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.53.21.2 complementary

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)

- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.53.22 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the

input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.53.22.1 interaction_order

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.53.22.2 drop_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
    sample_type lhs
    samples = 100
    variance_based_decomp
    drop_tolerance = 0.001
```

7.2.53.23 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.53.24 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)

- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.53.25 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.53.25.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) annotated.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081      0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081      0.760045
...
```

7.2.53.25.1.1 header

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.53.25.1.2 `eval_id`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.53.25.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.53.25.2 annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.53.25.3 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.53.25.4 active_only

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.53.26 export_approx_points_file

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: export_points_file

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.53.26.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002      0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.53.26.1.1 header

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.53.26.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.53.26.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.53.26.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.53.26.3 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.53.27 seed_sequence

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For $\$L\$$ levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 100 50 40 30 20
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.53.28 `fixed_seed`

- [Keywords Area](#)

- [method](#)
- [multilevel_polynomial_chaos](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.53.29 model_pointer

- [Keywords Area](#)
- [method](#)
- [multilevel_polynomial_chaos](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.54 stoch_collocation

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)

Uncertainty quantification with stochastic collocation

Specification

Alias: nond_stoch_collocation

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates
	Required (<i>Choose One</i>)	Interpolation Grid Type (Group 2)	quadrature_order	Order for tensor-products of Gaussian quadrature rules

			sparse_grid_level	Level to use in sparse grid integration or interpolation
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional		sample_type	Selection of sampling strategy
	Optional		rng	Selection of a random number generator
	Optional		probability_refinement	Allow refinement of probability and generalized reliability results using importance sampling

	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate

	Optional	export_approx_points_file	Output file for surrogate model value evaluations
	Optional	seed	Seed of the random number generator
	Optional	fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional	model_pointer	Identifier for model block to be used by a method

Description

Stochastic collocation is a general framework for approximate representation of random response functions in terms of finite-dimensional interpolation bases.

The stochastic collocation (SC) method is very similar to [polynomial_chaos](#), with the key difference that the orthogonal polynomial basis functions are replaced with interpolation polynomial bases. The interpolation polynomials may be either local or global and either value-based or gradient-enhanced. In the local case, valued-based are piecewise linear splines and gradient-enhanced are piecewise cubic splines, and in the global case, valued-based are Lagrange interpolants and gradient-enhanced are Hermite interpolants. A value-based expansion takes the form

$$R = \sum_{i=1}^{N_p} r_i L_i(\xi)$$

where N_p is the total number of collocation points, r_i is a response value at the i^{th} collocation point, L_i is the i^{th} multidimensional interpolation polynomial, and ξ is a vector of standardized random variables.

Thus, in PCE, one forms coefficients for known orthogonal polynomial basis functions, whereas SC forms multidimensional interpolation functions for known coefficients.

Basis polynomial family (Group 2)

In addition to the [askey](#) and [wiener](#) basis types also supported by [polynomial_chaos](#), SC supports the option of `piecewise` local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis options provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space. The global gradient-enhanced interpolants (Hermite interpolation polynomials) are also restricted to uniform or transformed uniform random variables (due to the need to compute collocation weights by integration of the basis polynomials) and share the variable support shown in [variable_support](#) for Piecewise SE. Due to numerical instability in these high-order basis polynomials, they are deactivated by default but can be activated by developers using a compile-time switch.

Interpolation grid type (Group 3)

To form the multidimensional interpolants L_i of the expansion, two options are provided.

1. interpolation on a tensor-product of Gaussian quadrature points (specified with `quadrature_order` and,

optionally, `dimension_preference` for anisotropic tensor grids). As for PCE, non-nested Gauss rules are employed by default, although the presence of `p_refinement` or `h_refinement` will result in default usage of nested rules for normal or uniform variables after any variable transformations have been applied (both defaults can be overridden using explicit `nested` or `non_nested` specifications).

- interpolation on a Smolyak sparse grid (specified with `sparse_grid_level` and, optionally, `dimension_preference` for anisotropic sparse grids) defined from Gaussian rules. As for sparse PCE, nested rules are employed unless overridden with the `non_nested` option, and the growth rules are restricted unless overridden by the `unrestricted` keyword.

Another distinguishing characteristic of stochastic collocation relative to [polynomial_chaos](#) is the ability to reformulate the interpolation problem from a `nodal` interpolation approach into a `hierarchical` formulation in which each new level of interpolation defines a set of incremental refinements (known as hierarchical surpluses) layered on top of the interpolants from previous levels. This formulation lends itself naturally to uniform or adaptive refinement strategies, since the hierarchical surpluses can be interpreted as error estimates for the interpolant. Either global or local/piecewise interpolants in either value-based or gradient-enhanced approaches can be formulated using `hierarchical` interpolation. The primary restriction for the hierarchical case is that it currently requires a sparse grid approach using nested quadrature rules (Genz-Keister, Gauss-Patterson, or Newton-Cotes for standard normals and standard uniforms in a transformed space: Askey, Wiener, or Piecewise settings may be required), although this restriction can be relaxed in the future. A selection of `hierarchical` interpolation will provide greater precision in the increments to mean, standard deviation, covariance, and reliability-based level mappings induced by a grid change within uniform or goal-oriented adaptive refinement approaches (see following section).

It is important to note that, while `quadrature_order` and `sparse_grid_level` are array inputs, only one scalar from these arrays is active at a time for a particular expansion estimation. These scalars can be augmented with a `dimension_preference` to support anisotropy across the random dimension set. The array inputs are present to support advanced use cases such as multifidelity UQ, where multiple grid resolutions can be employed.

Automated refinement type (Group 1)

Automated expansion refinement can be selected as either `p_refinement` or `h_refinement`, and either refinement specification can be either `uniform` or `dimension_adaptive`. The `dimension_adaptive` case can be further specified as either `sobol` or `generalized` (decay not supported). Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls. The `h_refinement` specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the `piecewise` specification option (it is not necessary to redundantly specify `piecewise` in the case of `h_refinement`). In future releases, the `hierarchical` interpolation approach will enable local refinement in addition to the current `uniform` and `dimension_adaptive` options.

Covariance type (Group 5)

These two keywords are used to specify how this method computes, stores, and outputs the covariance of the responses. In particular, the diagonal covariance option is provided for reducing post-processing overhead and output volume in high dimensional applications.

Active Variables

The default behavior is to form expansions over aleatory uncertain continuous variables. To form expansions over a broader set of variables, one needs to specify `active` followed by `state`, `epistemic`, `design`, or `all` in the variables specification block.

For continuous design, continuous state, and continuous epistemic uncertain variables included in the expansion, interpolation points for these dimensions are based on Gauss-Legendre rules if non-nested, Gauss-Patterson rules if nested, and Newton-Cotes points in the case of piecewise bases. Again, when probability integrals are evaluated, only the aleatory random variable domain is integrated, leaving behind a polynomial relationship between the statistics and the remaining design/state/epistemic variables.

Optional Keywords regarding method outputs

Each of these sampling specifications refer to sampling on the SC approximation for the purposes of generating approximate statistics.

- `sample_type`
- `samples`

- seed
- fixed_seed
- rng
- probability_refinement
- distribution
- reliability_levels
- response_levels
- probability_levels
- gen_reliability_levels

Since SC approximations are formed on structured grids, there should be no ambiguity with simulation sampling for generating the SC expansion.

When using the `probability_refinement` control, the number of refinement samples is not under the user's control (these evaluations are approximation-based, so management of this expense is less critical). This option allows for refinement of probability and generalized reliability results using importance sampling.

Multi-fidelity UQ

When using multifidelity UQ, the high fidelity expansion generated from combining the low fidelity and discrepancy expansions retains the polynomial form of the low fidelity expansion (only the coefficients are updated). Refer to [polynomial_chaos](#) for information on the multifidelity interpretation of array inputs for `quadrature_order` and `sparse_grid_level`.

Usage Tips

If n is small, then tensor-product Gaussian quadrature is again the preferred choice. For larger n , tensor-product quadrature quickly becomes too expensive and the sparse grid approach is preferred. For self-consistency in growth rates, nested rules employ restricted exponential growth (with the exception of the `dimension_adaptive p_refinement generalized` case) for consistency with the linear growth used for non-nested Gauss rules (integrand precision $i = 4l + 1$ for sparse grid level l and $i = 2m - 1$ for tensor grid order m).

Additional Resources

Dakota provides access to SC methods through the `NonDStochCollocation` class. Refer to the Uncertainty Quantification Capabilities chapter of the Users Manual [5] and the Stochastic Expansion Methods chapter of the Theory Manual [16] for additional information on the SC algorithm.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Integration and Expansion Moments](#)
- [Probability Density](#)
- [Level Mappings](#)
- [Variance-Based Decomposition \(Sobol' Indices\)](#)

Examples

```
method,
  stoch_collocation
    sparse_grid_level = 2
    samples = 10000 seed = 12347 rng rnum2
    response_levels = .1 1. 50. 100. 500. 1000.
    variance_based_decomp
```

Theory

As mentioned above, a value-based expansion takes the form

$$R = \sum_{i=1}^{N_p} r_i L_i(\xi)$$

The i^{th} interpolation polynomial assumes the value of 1 at the i^{th} collocation point and 0 at all other collocation points, involving either a global Lagrange polynomial basis or local piecewise splines. It is easy to see that the approximation reproduces the response values at the collocation points and interpolates between these values at other points. A gradient-enhanced expansion (selected via the `use_derivatives` keyword) involves both type 1 and type 2 basis functions as follows:

$$R = \sum_{i=1}^{N_p} [r_i H_i^{(1)}(\xi) + \sum_{j=1}^n \frac{dr_i}{d\xi_j} H_{ij}^{(2)}(\xi)]$$

where the i^{th} type 1 interpolant produces 1 for the value at the i^{th} collocation point, 0 for values at all other collocation points, and 0 for derivatives (when differentiated) at all collocation points, and the ij^{th} type 2 interpolant produces 0 for values at all collocation points, 1 for the j^{th} derivative component at the i^{th} collocation point, and 0 for the j^{th} derivative component at all other collocation points. Again, this expansion reproduces the response values at each of the collocation points, and when differentiated, also reproduces each component of the gradient at each of the collocation points. Since this technique includes the derivative interpolation explicitly, it eliminates issues with matrix ill-conditioning that can occur in the gradient-enhanced PCE approach based on regression. However, the calculation of high-order global polynomials with the desired interpolation properties can be similarly numerically challenging such that the use of local cubic splines is recommended due to numerical stability.

See Also

These keywords may also be of interest:

- [adaptive_sampling](#)
- [gpais](#)
- [local_reliability](#)
- [global_reliability](#)
- [sampling](#)
- [importance_sampling](#)
- [polynomial_chaos](#)

7.2.54.1 p_refinement

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	<code>uniform</code>	Refine an expansion uniformly in all dimensions.
			<code>dimension_adaptive</code>	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.54.1.1 uniform

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed.

7.2.54.1.2 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.54.1.2.1 sobol

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.54.1.2.2 generalized

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.54.2 h_refinement

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	h-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_-adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either `p_refinement` or `h_refinement`, and either refinement specification can be either `uniform` or `dimension_adaptive`. The `dimension_adaptive` case can be further specified as either `sobol` or `generalized` (decay not supported). Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls. The `h_refinement` specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify `piecewise` in the case of `h_refinement`). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current `uniform` and `dimension_adaptive` options.

7.2.54.2.1 uniform

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.54.2.2 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.
--	--	--	-----------------------------	---

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.54.2.2.1 sobol

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.54.2.2.2 generalized

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,  
  polynomial_chaos  
    sparse_grid_level = 3  
    dimension_adaptive p_refinement generalized  
    max_iterations = 20  
    convergence_tol = 1.e-4
```

7.2.54.2.3 local_adaptive

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.54.3 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive generalized  
  p_refinement  
  max_refinement_iterations = 20  
  convergence_tol = 1.e-4  
  sparse_grid_level = 1
```

7.2.54.4 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.54.5 `metric_scale`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	scale of refinement metric (Group 1)	relative	relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output

Usage Tips

Additional Discussion

7.2.54.5.1 `relative`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

Expected Output

Usage Tips

Additional Discussion7.2.54.5.2 `absolute`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior**Expected Output****Usage Tips****Additional Discussion**7.2.54.6 `quadrature_order`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		dimension_ - preference	A set of weights specifying the relative importance of each uncertain variable (dimension) Enforce use of nested quadrature rules if available
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of non-nested quadrature rules
			non_nested	

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`, where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and `method-multifidelity_stoch_collocation-quadrature_order_sequence`

7.2.54.6.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.54.6.2 nested

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.54.6.3 non_nested

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.54.7 sparse_grid_level

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation
		hierarchical	Employ a hierarchical sparse grid construction
Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.54.7.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.54.7.2 nodal

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sparse_grid_level](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.54.7.3 hierarchical

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sparse_grid_level](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.54.7.4 restricted

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.54.7.5 unrestricted

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)

- [sparse_grid_level](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.54.7.6 nested

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sparse_grid_level](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: `non_nested` unless automated refinement; sparse grids: `nested`

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.54.7.7 non_nested

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.54.8 piecewise

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.54.9 askey

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.54.10 wiener

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.54.11 use_derivatives

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.54.12 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.54.13 sample_type

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Sample Type (Group 1)	Dakota Keyword lhs	Dakota Keyword Description Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.54.13.1 lhs

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.54.13.2 random

- [Keywords Area](#)
- [method](#)

- [stoch_collocation](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.54.14 rng

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.54.14.1 mt19937

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.54.14.2 `rnum2`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```


7.2.54.15 probability_refinement

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: sample_refinement

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group Importance Sampling Approach (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.54.15.1 import

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.54.15.2 adapt_import

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [probability_refinement](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.54.15.3 mm_adapt_import

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.54.15.4 refinement_samples

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.54.16 `final_moments`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
```

```

samples = 50 seed = 1234
response_levels = 3.6e+11 1.2e+05 3.5e+05
final_moments central

```

Overriding the default to `none` as follows:

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```

allows associated nested model mappings to be simplified from:

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.

```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```

model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.

```

7.2.54.16.1 none

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```

method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none

```

7.2.54.16.2 standard

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.54.16.3 central

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.54.17 response_levels

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'
```



```

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06
4.0000000000e+05	4.5000000000e+05	1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
3.6000000000e+11	5.5000000000e-01		
4.0000000000e+11	3.8000000000e-01		
4.4000000000e+11	2.3000000000e-01		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
6.0000000000e+04	6.1000000000e-01		
6.5000000000e+04	2.9000000000e-01		
7.0000000000e+04	9.0000000000e-02		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
3.5000000000e+05	5.2000000000e-01		
4.0000000000e+05	9.0000000000e-02		
4.5000000000e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.54.17.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.54.17.2 compute

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
```

```

                                3.5e+05 4.0e+05 4.5e+05
compute reliabilities

```

7.2.54.17.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```

method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities

```

7.2.54.17.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)

- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.54.17.2.3 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.54.17.2.4 system

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system

			parallel	Aggregate response statistics assuming a parallel system
--	--	--	--------------------------	--

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.54.17.2.5 series

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.54.17.2.6 parallel

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.54.18 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.


```

method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		

```

4.0634975300e+11  3.3000000000e-01
5.4196114379e+11  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
4.6431154744e+04  1.0000000000e+00
5.6511827775e+04  8.0000000000e-01
6.1603813790e+04  5.0000000000e-01
7.8702465755e+04  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
2.3796737090e+05  1.0000000000e+00
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.4111498127e+05  0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.54.18.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.54.19 reliability_levels

- [Keywords Area](#)
- [method](#)

- [stoch_collocation](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.54.19.1 `num_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: reliability_levels evenly distributed among response functions

Description

See parent page

7.2.54.20 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.54.20.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.54.21 distribution

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Computes statistics according to cumulative functions

			complementary	Computes statistics according to complementary cumulative functions
--	--	--	-------------------------------	---

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.54.21.1 cumulative

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.54.21.2 complementary

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.54.22 variance_based_decomp

- [Keywords Area](#)

- [method](#)
- [stoch_collocation](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.54.22.1 interaction_order

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.54.22.2 drop_tolerance

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
    sample_type lhs
    samples = 100
    variance_based_decomp
    drop_tolerance = 0.001
```

7.2.54.23 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.54.24 full_covariance

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.54.25 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.54.25.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
--	-------------------	----------------------	----------------	----------------------------

	Optional	header	Enable header row in custom-annotated tabular file
	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1         0.0002  0.26           0.76
2              0.90009    1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991    1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.54.25.1.1 header

- [Keywords Area](#)
- [method](#)

- [stoch_collocation](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.54.25.1.2 `eval_id`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.54.25.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [import_approx_points_file](#)
- [custom_annotated](#)

- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.54.25.2 annotated

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated header eval_id`

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.54.25.3 freeform

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.54.25.4 active_only

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.54.26 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.54.26.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: `annotated` format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857  0.2601620081      0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081      0.760045
...
```

7.2.54.26.1.1 header

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.54.26.1.2 `eval_id`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.54.26.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.54.26.2 annotated

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.54.26.3 freeform

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.54.27 seed

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.54.28 fixed_seed

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.54.29 model_pointer

- [Keywords Area](#)
- [method](#)
- [stoch_collocation](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.55 multifidelity_stoch_collocation

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)

Multifidelity uncertainty quantification using stochastic collocation

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence

	Optional		metric_scale	define scaling of statistical metrics when adapting UQ surrogates
	Optional		statistics_mode	type of statistical metric roll-up for multifidelity UQ methods
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_emulation	Formulation for emulation of model discrepancies.
	Required (Choose One)	Interpolation Grid Type (Group 2)	quadrature_order_sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_sequence	Sequence of sparse grid levels used in a multi-stage expansion
	Optional (Choose One)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional		sample_type	Selection of sampling strategy
	Optional		rng	Selection of a random number generator
	Optional		probability_refinement	Allow refinement of probability and generalized reliability results using importance sampling
	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value

	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix
	Optional		import_approx_points_file	Filename for points at which to evaluate the PCE/SC surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		model_pointer	Identifier for model block to be used by a method

Description

As described in [stoch_collocation](#), stochastic collocation is a general framework for approximate representation of random response functions in terms of finite-dimensional interpolation bases, using interpolation polynomials that

may be either local or global and either value-based or gradient-enhanced.

In the multifidelity case, we decompose this interpolant expansion into several constituent expansions, one per model form or solution control level. In a bi-fidelity case with low-fidelity (LF) and high-fidelity (HF) models and an additive discrepancy approach, we have:

$$R = \sum_{i=1}^{N_p^{LF}} r_i^{LF} L_i(\xi) + \sum_{i=1}^{N_p^{HF}} \delta_i L_i(\xi)$$

where δ_i is a coefficient for the discrepancy expansion.

The same specification options are available as described in [stoch_collocation](#) with one key difference: the coefficient estimation inputs change from a scalar input for a single expansion to a *sequence* specification for a low-fidelity expansion followed by multiple discrepancy expansions.

To obtain the coefficients r_i and δ_i for each of the expansions, the following options are provided:

1. multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order_sequence`, and, optionally, `dimension_preference`).
2. multidimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level_sequence` and, optionally, `dimension_preference`)

It is important to note that, while `quadrature_order_sequence` and `sparse_grid_level_sequence` are array inputs, only one scalar from these arrays is active at a time for a particular expansion estimation. In order to specify anisotropy in resolution across the random variable set, a `dimension_preference` specification can be used to augment these scalar specifications.

Multifidelity UQ using SC requires that the model selected for iteration by the method specification is a multifidelity surrogate model (see [hierarchical](#)), which defines an `ordered_model_sequence` (see [hierarchical](#)). Two types of hierarchies are supported: (i) a hierarchy of model forms composed from more than one model within the `ordered_model_sequence`, or (ii) a hierarchy of discretization levels comprised from a single model within the `ordered_model_sequence` which in turn specifies a `solution_level_control` (see [solution_level_control](#)).

In both cases, an expansion will first be formed for the low fidelity model or coarse discretization, using the first value within the coefficient estimation sequence, along with any specified refinement strategy. Second, expansions are formed for one or more model discrepancies (the difference between response results if `additive_correction` or the ratio of results if `multiplicative_correction`), using all subsequent values in the coefficient estimation sequence (if the sequence does not provide a new value, then the previous value is reused) along with any specified refinement strategy. The number of discrepancy expansions is determined by the number of model forms or discretization levels in the hierarchy.

After formation and refinement of the constituent expansions, each of the expansions is combined (added or multiplied) into an expansion that approximates the high fidelity model, from which the final set of statistics are generated.

Additional Resources

Dakota provides access to multifidelity SC methods through the `NonDMultilevelStochCollocation` class. Refer to the Stochastic Expansion Methods chapter of the Theory Manual [16] for additional information on the Multifidelity SC algorithm.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Integration and Expansion Moments](#) (expansion moments only)
- [Probability Density](#)
- [Level Mappings](#)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

Examples

```
method,
  multifidelity_stoch_collocation
  model_pointer = 'HIERARCH'
  sparse_grid_level_sequence = 4 3 2

model,
  id_model = 'HIERARCH'
  surrogate_hierarchical
  ordered_model_fidelities = 'LF' 'MF' 'HF'
  correction_additive_zeroth_order
```

See Also

These keywords may also be of interest:

- [adaptive_sampling](#)
- [gpais](#)
- [local_reliability](#)
- [global_reliability](#)
- [sampling](#)
- [importance_sampling](#)
- [stoch_collocation](#)

7.2.55.1 p_refinement

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.55.1.1 uniform

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.55.1.2 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.55.1.2.1 sobol

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,  
    polynomial_chaos  
    sparse_grid_level = 3  
    dimension_adaptive p_refinement sobol  
    max_iterations = 20  
    convergence_tol = 1.e-4
```

7.2.55.1.2.2 generalized

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.55.2 h_refinement

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	h-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.

			dimension_-adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either `p_refinement` or `h_refinement`, and either refinement specification can be either `uniform` or `dimension_adaptive`. The `dimension_adaptive` case can be further specified as either `sobol` or `generalized` (decay not supported). Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls. The `h_refinement` specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify `piecewise` in the case of `h_refinement`). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current `uniform` and `dimension_adaptive` options.

7.2.55.2.1 uniform

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed.

7.2.55.2.2 `dimension_adaptive`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword	Dakota Keyword Description
			sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either `sobol` or `generalized` for stochastic collocation and either `sobol`, `decay`, or `generalized` for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.55.2.2.1 `sobol`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [h_refinement](#)

- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,  
    polynomial_chaos  
    sparse_grid_level = 3  
    dimension_adaptive p_refinement sobol  
    max_iterations = 20  
    convergence_tol = 1.e-4
```

7.2.55.2.2.2 generalized

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.55.2.3 local_adaptive

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.55.3 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive_generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.55.4 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)

- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.55.5 metric_scale

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [metric_scale](#)

define scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group scale of refinement metric (Group 1)	Dakota Keyword relative	Dakota Keyword Description relative scaling of statistical metrics when adapting UQ surrogates
			absolute	use of absolute statistical metrics when adapting UQ surrogates

Description

When adapting a UQ surrogate model, e.g., by change in response covariance matrix or level mappings, this option controls the use of scaling. The `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance. The `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

The default is `relative`.

Expected Output

Usage Tips

Additional Discussion

7.2.55.5.1 relative

- [Keywords Area](#)
- [method](#)

- [multifidelity_stoch_collocation](#)
- [metric_scale](#)
- [relative](#)

relative scaling of statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `relative` option indicates that the change is measured relative to the current reference value (which evolves during the refinement), such that the convergence tolerance controls the relative change.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.55.5.2 absolute

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [metric_scale](#)
- [absolute](#)

use of absolute statistical metrics when adapting UQ surrogates

Specification

Alias: none

Argument(s): none

Description

When adapting a UQ surrogate model, the `absolute` option indicates that there is no scaling such that the statistical change is compared directly to the convergence tolerance.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.55.6 `statistics_mode`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [statistics_mode](#)

type of statistical metric roll-up for multifidelity UQ methods

Specification

Alias: none

Argument(s): none

Default: `DEFAULT_EXPANSION_STATS`

Child Keywords:

	Required/Optional <i>Required(Choose One)</i>	Description of Group Statistics mode for refinement metric (Group 1)	Dakota Keyword active	Dakota Keyword Description use statistics for the active expansion within multifidelity refinement
			combined	use statistics for the combined expansion within multifidelity refinement

Description

When performing adaptive refinement in multifidelity UQ methods, one can adapt the expansion for a model level/fidelity based on its local statistics ([active](#)) or based on the statistics for the aggregate expansion ([combined](#)). The latter case may incur additional computational overhead, but ensures that the refinement occurs in the aggregate multifidelity context.

Default Behavior

Default for multifidelity methods is `combined` statistics.

Expected Output**Usage Tips****Additional Discussion**7.2.55.6.1 `active`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [statistics_mode](#)

- [active](#)

use statistics for the active expansion within multifidelity refinement

Specification

Alias: none

Argument(s): none

Description

When performing adaptive refinement in multifidelity UQ methods, this option adapts based on the statistics for the active model level/fidelity. This approach avoids some computational overhead in some cases by avoiding expansion roll-up.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.55.6.2 [combined](#)

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [statistics_mode](#)
- [combined](#)

use statistics for the combined expansion within multifidelity refinement

Specification

Alias: none

Argument(s): none

Description

When performing adaptive refinement in multifidelity UQ methods, this option adapts based on the statistics for the aggregate multifidelity expansion. This approach may incur additional computational overhead, but ensures that the refinement occurs in the aggregate multifidelity context.

Default Behavior

Expected Output

Usage Tips

Additional Discussion

7.2.55.7 allocation_control

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity stochastic collocation

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.55.7.1 greedy

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity stochastic collocation

Specification

Alias: none

Argument(s): none

Description

Multifidelity stochastic collocation supports greedy refinement strategies using tensor and sparse grids for both nodal and hierarchical collocation approaches. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within an integrated competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity stochastic collocation using nodal interpolation starts from a zeroth-order expansion (a constant) for each level, and generates uniform candidate refinements for each level that are competed in a greedy competition. The number of new samples for the incremented candidate expansion order is determined from the quadrature rules of the new sparse grid level. In this case, the number of candidates for each level is limited to one uniform refinement of the current sparse grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
    nodal
    allocation_control greedy
    p_refinement uniform
    sparse_grid_level_sequence = 0 unrestricted
    convergence_tolerance 1.e-3
```

The next example employs generalized sparse grids and hierarchical interpolation. Each level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all levels is competed within a unified greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each level and the number of candidates grows rapidly with random dimension and grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
    hierarchical
    allocation_control greedy
    p_refinement dimension_adaptive generalized
    sparse_grid_level_sequence = 0 unrestricted
    convergence_tolerance 1.e-8
```

7.2.55.8 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Discrepancy Emulation Approach (Group 1)	distinct	Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
    multilevel_polynomial_chaos
    expansion_order_sequence = 2
    collocation_ratio = .9
    orthogonal_matching_pursuit
    discrepancy_emulation recursive
```

7.2.55.8.1 `distinct`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.55.8.2 recursive

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.55.9 quadrature_order_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.55.9.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.55.9.2 nested

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.55.9.3 non_nested

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.55.10 sparse_grid_level_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation
			hierarchical	Employ a hierarchical sparse grid construction
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.55.10.1 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.55.10.2 `nodal`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)

- [sparse_grid_level_sequence](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.55.10.3 hierarchical

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sparse_grid_level_sequence](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.55.10.4 restricted

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sparse_grid_level_sequence](#)

- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.55.10.5 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sparse_grid_level_sequence](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed

sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.55.10.6 nested

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sparse_grid_level_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.55.10.7 non_nested

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.55.11 piecewise

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.55.12 askey

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.55.13 wiener

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.55.14 use_derivatives

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.55.15 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: samples

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.55.16 sample_type

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Sample Type (Group 1)	Dakota Keyword lhs	Dakota Keyword Description Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
```

```
samples = 20
seed = 83921
```

7.2.55.16.1 lhs

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
  samples = 20
```

7.2.55.16.2 random

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.55.17 rng

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (`mt19937`)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.55.17.1 mt19937

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
```



```
samples = 10
seed = 98765
rng mt19937
```

7.2.55.17.2 rnum2

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.55.18 probability_refinement

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: sample_refinement

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Importance Sampling Approach (Group 1)	import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.55.18.1 import

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.55.18.2 `adapt_import`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [probability_refinement](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.55.18.3 `mm_adapt_import`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more

probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.55.18.4 refinement_samples

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.55.19 final_moments

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required(Choose One)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.55.19.1 none

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,  
  sampling  
    samples = 50 seed = 1234  
    response_levels = 3.6e+11 1.2e+05 3.5e+05  
    final_moments none
```

7.2.55.19.2 standard

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.55.19.3 central

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.55.20 response_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_response_levels</code>	Number of values at which to estimate desired statistics for each response
	Optional		<code>compute</code>	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num-`

`_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06

```
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06
```

Level mappings for each response function:

```
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.6000000000e+11  5.5000000000e-01
  4.0000000000e+11  3.8000000000e-01
  4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  6.0000000000e+04  6.1000000000e-01
  6.5000000000e+04  2.9000000000e-01
  7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.5000000000e+05  5.2000000000e-01
  4.0000000000e+05  9.0000000000e-02
  4.5000000000e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.55.20.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.55.20.2 compute

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.55.20.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.55.20.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.55.20.2.3 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.55.20.2.4 system

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.55.20.2.5 series

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.55.20.2.6 `parallel`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.55.21 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds    = 199.3, 474.63
  upper_bounds    = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs       = 3 4
  abscissas       = 5 8 10 .1 .2 .3 .4
  counts          = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs       = 2
  abscissas       = 3 4
  counts          = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
```

```
no_gradients
no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.55.21.1 num_probability_levels

- [Keywords Area](#)
- [method](#)

- [multifidelity_stoch_collocation](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.55.22 `reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.55.22.1 num_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.55.23 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.55.23.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.55.24 `distribution`

- [Keywords Area](#)
- [method](#)

- [multifidelity_stoch_collocation](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.55.24.1 cumulative

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)

- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.55.24.2 complementary

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.55.25 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into main, interaction, and total effects

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interaction_order	Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

	Optional	<code>drop_tolerance</code>	Suppresses output of sensitivity indices with values lower than this tolerance
--	-----------------	-----------------------------	--

Description

Dakota can calculate sensitivity indices through variance-based decomposition using the keyword `variance_based_decomp`. This approach decomposes main, interaction, and total effects in order to identify the most important variables and combinations of variables in contributing to the variance of output quantities of interest.

Default Behavior

Because of processing overhead and output volume, `variance_based_decomp` is inactive by default, unless required for dimension-adaptive refinement using Sobol' indices.

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects, total effects, and any interaction effects will be reported. Each of these effects represents the percent contribution to the variance in the model response, where main effects include the aggregated set of *univariate* terms for each individual variable, interaction effects represent the set of *mixed* terms (the complement of the univariate set), and total effects represent the *complete* set of terms (univariate and mixed) that contain each individual variable. The aggregated set of main and interaction sensitivity indices will sum to one, whereas the sum of total effects sensitivity indices will be greater than one due to redundant counting of mixed terms.

Usage Tips

An important consideration is that the number of possible interaction terms grows exponentially with dimension and expansion order. To mitigate this, both in terms of compute time and output volume, possible interaction effects are suppressed whenever no contributions are present due to the particular form of an expansion. In addition, the `interaction_order` and `drop_tolerance` controls can further limit the computational and output requirements.

Examples

```
method,
  polynomial_chaos # or stoch_collocation
  sparse_grid_level = 3
  variance_based_decomp interaction_order = 2
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [90].

7.2.55.25.1 `interaction_order`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)

- [variance_based_decomp](#)
- [interaction_order](#)

Specify the maximum number of variables allowed in an interaction when reporting interaction metrics.

Specification

Alias: none

Argument(s): INTEGER

Default: Unrestricted (VBD includes all interaction orders present in the expansion)

Description

The `interaction_order` option has been added to allow suppression of higher-order interactions, since the output volume (and memory and compute consumption) of these results could be extensive for high dimensional problems (note: the previous `univariate_effects` specification is equivalent to `interaction_order = 1` in the current specification). Similar to suppression of interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to again save compute and memory resources and reduce output volume)

7.2.55.25.2 `drop_tolerance`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,  
  sampling  
  sample_type lhs  
  samples = 100  
  variance_based_decomp  
  drop_tolerance = 0.001
```

7.2.55.26 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: diagonal_covariance for response vector > 10; else full_covariance

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.55.27 full_covariance

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.55.28 import_approx_points_file

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [import_approx_points_file](#)

Filename for points at which to evaluate the PCE/SC surrogate

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Default Behavior No import of points at which to evaluate the surrogate.

Expected Output The PCE/SC surrogate model will be evaluated at the list of points (input variable values) provided in the file and results tabulated and/or statistics computed at them, depending on the method context.

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_approx_points_file = 'import.mcmc_annot.dat'
    annotated
```

7.2.55.28.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)

- [import_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.55.28.1.1 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.55.28.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.55.28.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [import_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.55.28.2 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [import_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.55.28.3 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [import_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.55.28.4 active_only

- [Keywords Area](#)

- [method](#)
- [multifidelity_stoch_collocation](#)
- [import_approx_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.55.29 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.55.29.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.55.29.1.1 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.55.29.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.55.29.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.55.29.2 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.55.29.3 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.55.30 `seed_sequence`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For L levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 100 50 40 30 20
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.55.31 `fixed_seed`

- [Keywords Area](#)

- [method](#)
- [multifidelity_stoch_collocation](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.55.32 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [multifidelity_stoch_collocation](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.56 sampling

- [Keywords Area](#)
- [method](#)
- [sampling](#)

Randomly samples variables according to their distributions

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)
- [sampling](#)

Specification

Alias: nond_sampling

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		samples	Number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		sample_type	Selection of sampling strategy
	Optional		refinement_-samples	Performs an incremental Latin Hypercube Sampling (LHS) study

	Optional	d_optimal	Generate a D-optimal sampling design
	Optional	variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into contributions from variables
	Optional	backfill	Ensures that the samples of discrete variables with finite support are unique
	Optional	principal_components	Activates principal components analysis of the response matrix of N samples * L responses.
	Optional	wilks	Number of samples for random sampling using Wilks statistics
	Optional	final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	reliability_levels	Specify reliability levels at which the response values will be estimated

	Optional	gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions
	Optional	rng	Selection of a random number generator
	Optional	model_pointer	Identifier for model block to be used by a method

Description

This method generates parameter values by drawing samples from the specified uncertain variable probability distributions. The computational model is executed over all generated parameter values to compute the responses for which statistics are computed. The statistics support sensitivity analysis and uncertainty quantification.

Default Behavior

By default, `sampling` methods operate on aleatory and epistemic uncertain variables. The types of variables can be restricted or expanded (to include design or state variables) through use of the `active` keyword in the `variables` block in the Dakota input file. If continuous design and/or state variables are designated as active, the sampling algorithm will treat them as parameters with uniform probability distributions between their upper and lower bounds. Refer to [variable_support](#) for additional information on supported variable types, with and without correlation.

The following keywords change how the samples are selected:

- `sample_type`
- `fixed_seed`
- `rng`
- `samples`
- `seed`
- `variance_based_decomp`

Expected Outputs

As a default, Dakota provides correlation analyses when running LHS. Correlation tables are printed with the simple, partial, and rank correlations between inputs and outputs. These can be useful to get a quick sense of how correlated the inputs are to each other, and how correlated various outputs are to inputs. `variance_based_decomp` is used to request more sensitivity information, with additional cost.

Additional statistics can be computed from the samples using the following keywords:

- `response_levels`
- `reliability_levels`
- `probability_levels`

- `gen_reliability_levels`

`response_levels` computes statistics at the specified response value. The other three allow the specification of the statistic value, and will estimate the corresponding response value.

`distribution` is used to specify whether the statistic values are from cumulative or complementary cumulative functions.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- When `variance_based_decomp` is enabled
 - [Variance-Based Decomposition \(Sobol' Indices\)](#)
- For aleatory UQ studies
 - [Probability Density](#)
 - [Level Mappings](#)
 - [Sampling Moments](#)
 - [Correlations](#)
- For epistemic UQ studies
 - [Extreme Responses](#)
 - [Correlations](#)

Usage Tips

`sampling` is a robust approach to doing sensitivity analysis and uncertainty quantification that can be applied to any problem. It requires more simulations than newer, advanced methods. Thus, an alternative may be preferable if the simulation is computationally expensive.

Examples

```
# tested on Dakota 6.0 on 140501

environment
  tabular_data
    tabular_data_file = 'Sampling_basic.dat'

method
  sampling
  sample_type lhs
  samples = 20

model
  single

variables
  active uncertain
  uniform_uncertain = 2
  descriptors = 'input1'      'input2'
  lower_bounds = -2.0      -2.0
  upper_bounds =  2.0      2.0
  continuous_state = 1
  descriptors = 'constant1'
  initial_state = 100

interface
  analysis_drivers 'text_book'
  fork
```

```
responses
  response_functions = 1
  no_gradients
  no_hessians
```

This example illustrates a basic sampling Dakota input file.

- LHS is used instead of purely random sampling.
- The default random number generator is used.
- Without a `seed` specified, this will not be reproducible
- In the `variables` block, two types of variables are used
- Only the uncertain variables are varied, this is the default behavior, and is also specified by the `active` keyword, w/ the `uncertain` option

See Also

These keywords may also be of interest:

- [active](#)

FAQ

Q: Do I need to keep the LHS* and S4 files? **A:** No

7.2.56.1 samples

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: `initial_samples`

Argument(s): INTEGER

Default: 0

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10 \cdot \text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N \cdot (\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.56.2 seed

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.56.3 `fixed_seed`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.56.4 `sample_type`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: lhs

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sample Type (Group 1)	lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables
			incremental_lhs	(Deprecated keyword) Augments an existing Latin Hypercube Sampling (LHS) study
			incremental_random	(Deprecated keyword) Augments an existing random sampling study

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.56.4.1 lhs

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.56.4.2 random

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.56.4.3 incremental_lhs

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [sample_type](#)
- [incremental_lhs](#)

(Deprecated keyword) Augments an existing Latin Hypercube Sampling (LHS) study

Specification

Alias: none

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

This keyword is deprecated. Instead specify `sample_type lhs` with `refinement_samples`.

7.2.56.4.4 incremental_random

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [sample_type](#)
- [incremental_random](#)

(Deprecated keyword) Augments an existing random sampling study

Specification

Alias: none

Argument(s): none

Description

This keyword is deprecated. Instead specify `sample_type random` with `refinement_samples`.

7.2.56.5 `refinement_samples`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [refinement_samples](#)

Performs an incremental Latin Hypercube Sampling (LHS) study

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Use of `refinement_samples` replaces the deprecated `sample_type incremental_lhs` and `sample_type incremental_random`.

An incremental random sampling approach will successively add samples to an initial or existing random sampling study according to the sequence of `refinement_samples`. Dakota reports statistics (mean, variance, percentiles, etc) separately for the initial samples and for each `refinement_samples` increment at the end of the study. For an LHS design, the number of `refinement_samples` in each refinement level must result in twice the number of previous samples. For `sample_type random`, there is no constraint on the number of samples that can be added.

Often, this approach is used when you have an initial study with sample size N_1 and you want to perform an additional N_1 samples. The initial N_1 samples may be contained in a Dakota restart file so only N_1 (instead of $2 \times N_1$) potentially expensive function evaluations will be performed.

This process can be extended by repeatedly increasing (for LHS: doubling) the `refinement_samples`:

```
method
  sampling
    seed = 1337
    samples = 50
    refinement_samples = 50 100 200 400 800
```

Usage Tips

The incremental approach is useful if it is uncertain how many simulations can be completed within available time.

See the examples below and the [Usage](#) and [Restarting Dakota Studies](#) pages.

Examples

Suppose an initial study is conducted using `sample_type lhs` with `samples = 50`. A follow-on study uses a new input file where `samples = 50`, and `refinement_samples = 50`, resulting in 50 reused samples (from restart) and 50 new random samples. The 50 new samples will be combined with the 50 previous samples to generate a combined sample of size 100 for the analysis.

One way to ensure the restart file is saved is to specify a non-default name, via a command line option:

```
dakota -input LHS_50.in -write_restart LHS_50.rst
```

which uses the input file:

```
# LHS_50.in

environment
  tabular_data
    tabular_data_file = 'LHS_50.dat'

method
  sampling
    seed = 1337
    sample_type lhs
    samples = 50

model
  single

variables
  uniform_uncertain = 2
  descriptors = 'input1' 'input2'
  lower_bounds = -2.0 -2.0
  upper_bounds = 2.0 2.0

interface
  analysis_drivers 'text_book'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians
```

and the restart file is written to LHS_50.rst.

Then an incremental LHS study can be run with:

```
dakota -input LHS_100.in -read_restart LHS_50.rst -write_restart LHS_100.rst
```

where LHS_100.in is shown below, and LHS_50.rst is the restart file containing the results of the previous LHS study. In the example input files for the initial and incremental studies, the values for `seed` match. This ensures that the initial 50 samples generated in both runs are the same.

```
# LHS_100.in

environment
  tabular_data
    tabular_data_file = 'LHS_100.dat'

method
  sampling
    seed = 1337
    sample_type lhs
    samples = 50
    refinement_samples = 50

model
  single

variables
  uniform_uncertain = 2
  descriptors = 'input1' 'input2'
  lower_bounds = -2.0 -2.0
  upper_bounds = 2.0 2.0

interface
  analysis_drivers 'text_book'
  fork

responses
```

```

response_functions = 1
no_gradients
no_hessians

```

The user will get 50 new LHS samples which maintain both the correlation and stratification of the original LHS sample. The new samples will be combined with the original samples to generate a combined sample of size 100.

7.2.56.6 d_optimal

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [d_optimal](#)

Generate a D-optimal sampling design

Specification

Alias: none

Argument(s): none

Default: off

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Design Strategy (Group 1)	Dakota Keyword candidate_designs	Dakota Keyword Description Number of candidate sampling designs from which to select the most D-optimal
			leja_oversample_ratio	Oversampling ratio for generating candidate point set

Description

This option will generate a sampling design that is approximately determinant-optimal (D-optimal) by downselecting from a set of candidate sample points.

Default Behavior

If not specified, a standard sampling design (MC or LHS) will be generated. When `d_optimal` is specified, 100 candidate designs will be generated and the most D-optimal will be selected.

Usage Tips

D-optimal designs are only supported for [aleatory_uncertain_variables](#). The default candidate-based D-optimal strategy works for all submethods except incremental LHS (`lhs` with `refinement_samples`). The Leja sampling option only works for continuous variables, and when used with LHS designs, the candidates point set will be Latin, but the final design will not be.

Examples

```
method
  sampling
    sample_type random
    samples = 20
    d_optimal
```

7.2.56.6.1 candidate_designs

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [d_optimal](#)
- [candidate_designs](#)

Number of candidate sampling designs from which to select the most D-optimal

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Dakota will generate `candidate_designs` using the specified sampling strategy and select the one that is more determinant-optimal (D-optimal)

Examples

```
method
  sampling
    sample_type random
    samples = 20
    d_optimal
    candidate_designs = 500
```

7.2.56.6.2 leja_oversample_ratio

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [d_optimal](#)
- [leja_oversample_ratio](#)

Oversampling ratio for generating candidate point set

Specification

Alias: none

Argument(s): REAL

Default: 10.0

Description

When generating a D-Optimal point set of size N, the `oversample_ratio` R controls the number of candidate points R x N from which the D-Optimal points are chosen.

Default Behavior

The default when not specified is to perform candidate-based sample design selection (non-Leja)

Examples

```
method
  sampling
    sample_type random
    samples = 20
    d_optimal
      oversample_ratio = 2.0
```

7.2.56.7 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into contributions from variables

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance based decomposition using the keyword `variance_based_decomp`. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

Default Behavior

Because of the computational cost, `variance_based_decomp` is turned off as a default.

If the user specified a number of samples, N, and a number of nondeterministic variables, M, variance-based decomposition requires the evaluation of $N*(M+2)$ samples. **Note that specifying this keyword will increase the number of function evaluations above the number requested with the `samples` keyword since replicated sets of sample values are evaluated.**

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response

Usage Tips

To obtain sensitivity indices that are reasonably accurate, we recommend that N, the number of samples, be at least one hundred and preferably several hundred or thousands.

Examples

```
method,
  sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76] : "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [76] and [90].

7.2.56.7.1 `drop_tolerance`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

7.2.56.8 backfill

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [backfill](#)

Ensures that the samples of discrete variables with finite support are unique

Specification

Alias: none

Argument(s): none

Description

Traditional LHS can generate replicate samples when applied to discrete variables. This keyword enforces uniqueness, which is determined only over the set of discrete variables with finite support. This allows one to generate LHS for a mixed set of continuous and discrete variables whilst still enforcing that the set of discrete LHS components of all the samples are unique.

Default Behavior

Uniqueness of samples over discrete variables is not enforced.

Usage Tips

Uniqueness can be useful when applying discrete LHS to simulations without noise.

Examples

```
method,
  sampling
    samples = 12
    seed = 123456
    sample_type lhs backfill

variables,
  active all
  uniform_uncertain = 1
  lower_bounds = 0.
  upper_bounds = 1.
  descriptors = 'continuous-uniform'

  discrete_uncertain_set
    integer = 1
    elements_per_variable = 4
    elements 1 3 5 7
    descriptors = 'design-set-int'
    real = 1
    initial_point = 0.50
    set_values = 0.25 0.50 0.75 1.00
    descriptors = 'design-set-real'

interface,
  direct analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

See Also

These keywords may also be of interest:

- [lhs](#)

7.2.56.9 principal_components

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [principal_components](#)

Activates principal components analysis of the response matrix of N samples * L responses.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>percent_variance_explained</code>	Specifies the number of components to retain to explain the specified percent variance.

Description

Dakota can calculate the principal components of the response matrix of N samples * L responses using the keyword `principal_components`. Principal components analysis (PCA) is a data reduction method. The Dakota implementation is under active development: the PCA capability may ultimately be specified elsewhere or used in different ways. For now, it is performed as a post-processing analysis based on a set of Latin Hypercube samples.

We now have field responses in Dakota. PCA is an initial approach in Dakota to analyze and represent the field data. Specifically, if we have a sample ensemble of field data responses, we want to identify the principal components responsible for the spread of that data. Then, we can generate a surrogate model by representing the overall response as weighted sum of M principal components, where the weights will be determined by GPs which are a function of the input uncertain variables. This reduced form then can be used for sensitivity analysis, calibration, etc.

The steps involved when one specifies `principal_components` in Dakota are as follows:

- Create an LHS input sample based on the uncertain variable specification and run the user-specified model at the LHS points to compute the field responses. For notation purposes, there are d input parameters, N samples, and the field length is L .
- Perform PCA on the covariance matrix of the data set from the previous step. This is done by first centering the data (e.g. subtracting the mean of each column from that column) and performing a singular value decomposition on the covariance matrix of the centered data. The eigenvectors of the covariance matrix correspond to the principal components.
- Identify M principal components based on the percentage of variance explained. There is an optional keyword for `principal_components` called `percent_variance_explained`, which is a threshold that determines the number of components that are retained to explain at least that amount of variance. For example, if the user specifies `percent_variance_explained = 0.99`, the number of components that accounts for at least 99 percent of the variance in the responses will be retained. The default for this percentage is 0.95. In many applications, only a few principal components explain the majority of the variance, resulting in significant data reduction.
- Use the principal components in a predictive sense, by constructing a prediction approximation. The basis functions for this approximation are the principal components. The coefficients of the bases are obtained by constructing GP surrogates for the factor scores of the M principal components. The GP surrogates will be functions of the uncertain inputs. The idea is that we have just performed PCA on (for example) the covariance matrix of 100 samples. Typically, those 100 samples will be generated by sampling over some d uncertain input parameters denoted by u , so there should be a mapping from u to the field data, specifically to the loading coefficients and the factor scores. Currently, the final item printed from a Principal Components Analysis in Dakota is a set of prediction samples based on this prediction approximation or surrogate model that relies on the principal components.

Default Behavior

`principal_components` is turned off as a default. It may be used with either scalar responses or field responses, but it is intended to be used with large field responses as a data reduction method. For example, typically we expect the number of LHS samples, N , to be less than the number of field responses, L (e.g. if there is one field, the number of responses values is the length of that field).

Expected Outputs

When `principal_components` is specified, the number of significant principal components is printed along with the predictions based on the principal components. If `output debug` is specified, additional information is printed, including the original response matrix, the centered data, the principal components, and the factor scores.

Usage Tips

This is a preliminary capability that is undergoing active development. Please contact the Dakota developers team if you have problems with using this capability or want to suggest additional features.

Examples

```
method,
  sampling
  sample_type lhs
  samples = 100
  principal_components
  percent_variance_explained = 0.98
```

Theory

There is an extensive statistical literature available on PCA. We recommend that the interested user peruse some of this in using the PCA capability.

7.2.56.9.1 percent_variance_explained

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [principal_components](#)
- [percent_variance_explained](#)

Specifies the number of components to retain to explain the specified percent variance.

Specification

Alias: none

Argument(s): REAL

Description

Dakota can calculate the principal components of the response matrix of N samples * L responses using the keyword `principal_components`. Principal components analysis (PCA) is a data reduction method. `percent_variance_explained` is a threshold that determines the number of components that are retained to explain at least that amount of variance. For example, if the user specifies `percent_variance_explained = 0.99`, the number of components that accounts for at least 99 percent of the variance in the responses will be retained. The default for this percentage is 0.95. In many applications, only a few principal components explain the majority of the variance, resulting in significant data reduction.

Expected Outputs

Usage Tips `percent_variance_explained` should be a real number between 0.0 and 1.0. Typically, it will be between 0.9 and 1.0.

Examples

```
method,
  sampling
  sample_type lhs
  samples = 100
  principal_components
  percent_variance_explained = 0.98
```

Theory

There is an extensive statistical literature available on PCA. We recommend that the interested user peruse some of this in using the PCA capability.

7.2.56.10 wilks

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [wilks](#)

Number of samples for random sampling using Wilks statistics

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		order	The order of the statistics to use when determining sample sizes for random sampling using Wilks order statistics.
	Optional		confidence_level	The confidence level to be used when determining sample sizes for random sampling using Wilks order statistics.

	Optional	<code>one_sided_lower</code>	Specifies one-sided lower portion order statistics to be used when determining sample sizes for random sampling using Wilks order statistics.
	Optional	<code>one_sided_upper</code>	Specifies one-sided upper portion order statistics to be used when determining sample sizes for random sampling using Wilks order statistics.
	Optional	<code>two_sided</code>	Specifies two-sided order statistics (an interval) to be used when determining sample sizes for random sampling using Wilks order statistics.

Description

The `wilks` keyword is used to compute the number of samples to execute for a random sampling study using Wilks statistics [91] and [68]. In contrast to most sampling studies where the user specifies the number of samples in advance, Wilks determines the number of samples to run to achieve a particular objective. Specifically, Wilks statistics specify a probability level, alpha, and confidence level, beta, and determines the minimum number of samples required such that there is beta% confidence that the alpha*100 percentile of the uncertain distribution on model output will fall below the actual alpha*100 percentile given by the sample when outputs are ordered from smallest to largest. Statistics can also be either `one_side` or `two_sided` with the former reflecting a statement about uppermost sample output and the latter reflecting both the smallest and largest sample outputs. Finally, the order of the statistics can be increased to higher order such that the statement concerning probability level and confidence level applies to the uppermost M outputs (for one-sided M-order Wilks) or the lowest M and uppermost M outputs (for two-sided M-order Wilks).

Default Behavior

By default, Wilks statistics are computed using one-sided first-order order statistics with a 95% confidence interval (beta) and 95% probability (alpha). This results in a sample size of 59.

Usage Tips

Wilks sample sizes apply to model outputs considered one-at-a-time. Joint variation among multiple outputs requires a generalization of the Wilks approach and is not supported in Dakota at this time.

When more than one probability level is specified, the largest sample size will be performed and used to subsample for the lower probability levels.

Examples

```
method
  sampling
    sample_type random
```

```
wilks
  probability_levels = 0.75 0.8 0.95 0.99
  confidence_level 0.99
  two_sided
  order 2
```

7.2.56.10.1 order

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [wilks](#)
- [order](#)

The order of the statistics to use when determining sample sizes for random sampling using Wilks order statistics.

Specification

Alias: none

Argument(s): INTEGER

Description

Default Behavior

The default order is 1.

Examples

```
method
  sampling
  sample_type random
  wilks
  order 2
```

See Also

These keywords may also be of interest:

- [wilks](#)

7.2.56.10.2 confidence_level

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [wilks](#)
- [confidence_level](#)

The confidence level to be used when determining sample sizes for random sampling using Wilks order statistics.

Specification

Alias: none

Argument(s): REAL

Description

Default Behavior

A value of 0.95 (95%) is the default for both confidence level as well as probability level resulting in 59 samples for first-order one-sided statistics.

Examples

```
method
  sampling
    sample_type random
  wilks
    confidence_level 0.99
```

See Also

These keywords may also be of interest:

- [wilks](#) method-sampling-probability_levels

7.2.56.10.3 one_sided_lower

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [wilks](#)
- [one_sided_lower](#)

Specifies one-sided lower portion order statistics to be used when determining sample sizes for random sampling using Wilks order statistics.

Specification

Alias: none

Argument(s): none

Description

This option causes the sample size to be based on the lowermost N outputs for order N statistics (default N=1).

Examples

```
method
  sampling
    sample_type random
  wilks
    one_sided_lower
```

See Also

These keywords may also be of interest:

- [wilks](#)

7.2.56.10.4 one_sided_upper

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [wilks](#)
- [one_sided_upper](#)

Specifies one-sided upper portion order statistics to be used when determining sample sizes for random sampling using Wilks order statistics.

Specification

Alias: none

Argument(s): none

Description

This option causes the sample size to be based on the uppermost N outputs for order N statistics (default N=1).

Default Behavior This is the default for Wilks.

Examples

```
method
  sampling
    sample_type random
    wilks
    one_sided_upper
```

See Also

These keywords may also be of interest:

- [wilks](#)

7.2.56.10.5 two_sided

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [wilks](#)
- [two_sided](#)

Specifies two-sided order statistics (an interval) to be used when determining sample sizes for random sampling using Wilks order statistics.

Specification

Alias: none

Argument(s): none

Description

This option causes the sample size to be based on the interval defined by the lowest N and uppermost N outputs for order N statistics (default N=1).

Default Behavior The default for Wilks is `one_sided_upper`.

Examples

```
method
  sampling
    sample_type random
  wilks
    two_sided
```

See Also

These keywords may also be of interest:

- [wilks](#)

7.2.56.11 final_moments

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.

			<code>standard</code>	Output standardized moments and include them within the set of final statistics.
			<code>central</code>	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.56.11.1 none

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,  
  sampling  
    samples = 50 seed = 1234  
    response_levels = 3.6e+11 1.2e+05 3.5e+05  
    final_moments none
```

7.2.56.11.2 standard

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.56.11.3 central

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.56.12 response_levels

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_response_levels</code>	Number of values at which to estimate desired statistics for each response
	Optional		<code>compute</code>	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num-`

`_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06

```
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06
```

Level mappings for each response function:

```
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.6000000000e+11  5.5000000000e-01
  4.0000000000e+11  3.8000000000e-01
  4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  6.0000000000e+04  6.1000000000e-01
  6.5000000000e+04  2.9000000000e-01
  7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.5000000000e+05  5.2000000000e-01
  4.0000000000e+05  9.0000000000e-02
  4.5000000000e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.56.12.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.56.12.2 compute

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.56.12.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.56.12.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.56.12.2.3 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.56.12.2.4 system

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.56.12.2.5 series

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.56.12.2.6 `parallel`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.56.13 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
```

```
no_gradients
no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.56.13.1 num_probability_levels

- [Keywords Area](#)
- [method](#)

- [sampling](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.56.14 `reliability_levels`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.56.14.1 num_reliability_levels

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.56.15 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.56.15.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.56.16 `distribution`

- [Keywords Area](#)
- [method](#)

- [sampling](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.56.16.1 cumulative

- [Keywords Area](#)
- [method](#)
- [sampling](#)

- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.56.16.2 complementary

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.56.17 rng

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword	Dakota Keyword Description
			mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.56.17.1 mt19937

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.56.17.2 rnum2

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.56.18 model_pointer

- [Keywords Area](#)
- [method](#)
- [sampling](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.57 multilevel_sampling

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)

Multilevel sampling methods for UQ

Specification

Alias: multilevel_mc mlmc

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		pilot_samples	Initial set of samples for multilevel sampling methods.
	Optional		solution_mode	Solution mode for multi-level/multifidelity methods
	Optional		sample_type	Selection of sampling strategy
	Optional		export_sample_sequence	Enable export of multi-level/multifidelity sample sequences to individual files

	Optional	allocation_target	Allocation statistics/target for the MLMC sample allocation.
	Optional	qoi_aggregation	Aggregation strategy for the Qols statistics for problems with multiple responses in the MLMC algorithm
	Optional	convergence_tolerance	Stopping criterion based on relative error
	Optional	convergence_tolerance_type	Sets the type of the convergence tolerance. Can be absolute or relative. Default Behavior "relative"
	Optional	convergence_tolerance_target	Sets the target of the MLMC sample allocation, i.e. the constraint of the MLMC sample allocation problem Default Behavior "variance_constraint"
	Optional	max_iterations	Stopping criterion based on number of refinement iterations within the multilevel sample allocation
	Optional	max_function_evaluations	Stopping criterion based on maximum function evaluations
	Optional	final_statistics	Indicate the type of final statistics to be returned by a UQ method

	Optional	<code>rng</code>	Selection of a random number generator
	Optional	<code>model_pointer</code>	Identifier for model block to be used by a method

Description

An adaptive sampling method that utilizes multilevel relationships within a hierarchical surrogate model in order to improve efficiency through variance reduction.

In the case of a multilevel relationship, multilevel Monte Carlo methods are used to compute an optimal sample allocation per level.

Multilevel Monte Carlo

The Monte Carlo estimator for the mean is defined as

$$\mathbb{E}[Q] \equiv \hat{Q}^{MC} = \frac{1}{N} \sum_{i=1}^N Q^{(i)}$$

In a multilevel method with L levels, we replace this estimator with a telescoping sum:

$$\mathbb{E}[Q] \equiv \hat{Q}^{ML} = \sum_{l=0}^L \frac{1}{N_l} \sum_{i=1}^{N_l} (Q_l^{(i)} - Q_{l-1}^{(i)}) \equiv \sum_{l=0}^L \hat{Y}_l^{MC}$$

This decomposition forms discrepancies for each level greater than 0, seeking reduction in the variance of the discrepancy Y relative to the variance of the original response Q . The number of samples allocated for each level (N_l) is based on a total cost minimization procedure that incorporates the relative cost and observed variance for each of the Y_l .

Default Behavior

The `multilevel_sampling` method employs Monte Carlo sample sets by default, but this default can be overridden to use Latin hypercube sample sets using `sample_type lhs`.

Expected Output

The `multilevel_sampling` method reports estimates of the first four moments and a summary of the evaluations performed for each model fidelity and discretization level. The method does not support any level mappings (response, probability, reliability, generalized reliability) at this time.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Sampling Moments](#) (moments only, not confidence intervals)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

Usage Tips

The multilevel sampling method must be used in combination with a hierarchical model specification, and supports either a sequence of model forms or a sequence of discretization levels. For the former, each model form must provide a scalar `solution_level_cost` and for the latter, it is necessary to identify the variable string descriptor that controls the resolution levels using `solution_level_control` as well as the associated array of relative costs using `solution_level_cost`.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  max_iterations = 10
  convergence_tolerance = .001
```

specifies a multilevel Monte Carlo study in combination with the model identified by the HIERARCH pointer. This model specification provides a one-dimensional hierarchy, typically defined by a single model fidelity with multiple discretization levels, but may also be provided as multiple ordered model fidelities, each with a single (or default) discretization level. An example of the former (single model fidelity with multiple discretization levels) follows:

```
model,
  id_model = 'HIERARCH'
  surrogate hierarchical
  ordered_model_fidelities = 'SIM1'
  correction additive zeroth_order

model,
  id_model = 'SIM1'
  simulation
  solution_level_control = 'N_x'
  solution_level_cost = 630. 1260. 2100. 4200.
```

Refer to `dakota/test/dakota_uq*_mlmc.in` in the source distribution for additional examples.

See Also

These keywords may also be of interest:

- [multifidelity_sampling](#)
- [multilevel_multifidelity_sampling](#)

7.2.57.1 seed_sequence

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For $\$L\$$ levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 100 50 40 30 20
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.57.2 fixed_seed

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.57.3 pilot_samples

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [pilot_samples](#)

Initial set of samples for multilevel sampling methods.

Specification

Alias: `initial_samples`

Argument(s): INTEGERLIST

Description

The pilot sample provides initial estimates of variance and/or correlation within the first iteration of a multilevel and/or control variate approach.

Default Behavior

100 samples per model fidelity and/or discretization level.

Usage Tips

The number of specified values can be none (default values used for all fidelities and levels), one (all fidelities and levels use the same specified value), the number of discretization levels for every model (each model uses the

same discretization level profile), or the aggregate number of discretization levels for all models (samples for each discretization level are distinct for each model).

7.2.57.4 solution_mode

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [solution_mode](#)

Solution mode for multilevel/multifidelity methods

Specification

Alias: none

Argument(s): none

Default: online pilot

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Ensemble Sampling Solution Mode (Group 1)	online_pilot	Specify a solution mode that includes the pilot cost within the sample allocation logic
			offline_pilot	Specify a solution mode that excludes the pilot cost from sample allocation logic
			pilot_projection	Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. The pilot sample can be integrated according to one of three solution modes:

- `online_pilot`: include pilot sample cost within sample allocation (default)
- `offline_pilot`: treat pilot cost as offline and exclude from sample allocation

- `pilot_projection`: only perform the pilot sample and only estimate the expected performance based on a projection of the pilot statistics (useful for method selection)

Default Behavior Online pilot

7.2.57.4.1 `online_pilot`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [solution_mode](#)
- [online_pilot](#)

Specify a solution mode that includes the pilot cost within the sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as online cost and is included within the optimal sample allocation logic. This mode is the default.

Additional Discussion It is typically advantageous to start from a smaller pilot sample and then rely on the adaptive logic to increase the sample levels to the optimal level required to achieve the desired accuracy or enforce the prescribed budget. Over-estimation of the pilot, while supporting more robust estimation of correlations and variances, can hide the optimal profile and result in a non-optimal allocation of resources.

7.2.57.4.2 `offline_pilot`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [solution_mode](#)
- [offline_pilot](#)

Specify a solution mode that excludes the pilot cost from sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as offline cost and is excluded from the optimal sample allocation logic. This mode is typically used for generating reference performance using an "oracle" approach, i.e., specifying a large pilot sample in order to obtain best-case correlation or variance estimates prior to generating a sample profile.

7.2.57.4.3 pilot_projection

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [solution_mode](#)
- [pilot_projection](#)

Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, evaluations are limited to the pilot sample and these initial estimates are used to project the potential performance under the assumption that these estimates are accurate.

This mode is useful for comparing the projected performance of different estimators, which can inform method selection.

7.2.57.5 sample_type

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sample Type (Group 1)	lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.57.5.1 lhs

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.57.5.2 random

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.57.6 export_sample_sequence

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [export_sample_sequence](#)

Enable export of multilevel/multifidelity sample sequences to individual files

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Export sample sequences enabling file format customization
			annotated	Export sample sequences with descriptors
			freeform	Export sample sequences without heading descriptors

Description

When this option is active, separate output files are written for each unique sample increment and are tagged by algorithm type, simulation interface, iteration count, level count, and the number of samples as described below. The data content is comprised of the input variables only, without corresponding responses, as an intended use case is to support evaluation of these sample sets offline.

Default Behavior

If not specified, the `annotated` format is assumed.

Expected Output

Separate output files are generated according to the following format: `{ml/cv}_{interface_id}_{iteration_number}_{level_number}_{number_of_samples}.dat`.

With respect to the algorithm type, interface id, and level counter, the following definitions are employed:

- Multilevel MC with hierarchical model: `ml_` is pre-pended for all sample increments and sample sets are tagged with the interface id from the HF model.
- Control Variate MC with hierarchical model (deprecated): `cv_` is prepended for all sample increments, shared sample sets are tagged with the HF interface id, and LF-only refinements are tagged by the LF interface id.
- Multilevel-Control Variate MC with hierarchical model: employs a combination of exports from the two cases above. `ml_` is pre-pended for all HF sample increments and sample sets are tagged with the interface id from

the HF model; `cv_` is prepended for all LF increments and sample sets are tagged with the LF interface id. Note that the LF model shares the same `m1_` sample sets, but a redundant file is not created for this data.

- Non-hierarchical approaches (Multifidelity MC, Approximate Control Variate): `cv_` is prepended for all sample increments and sample sets are tagged with the corresponding model interface id. The level number corresponds to the index of the approximation model for LF increments and to the id of the truth model (number of LF approximations + 1) for shared increments.

With respect to iteration count, pilot samples are tagged with iteration 0.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  output_silent
  export_sample_sequence
```

results in enabling the sample output of sample increments for each level to individual files using the default annotated format.

The following variables block

```
variables,
  id_variables = 'LF_VARS'
  uniform_uncertain = 7
  lower_bounds = 7*-1.
  upper_bounds = 7* 1.
  descriptors 'u1' 'u2' 'u3' 'u4' 'u5' 'u6' 'u7'
  discrete_state_set
  integer = 2
  num_set_values = 4 1
  set_values = 5 15 30 60 # number of spatial coords
               3         # number of Fourier solution modes
  initial_state = 5 3
  descriptors 'N_x' 'N_mod'
```

illustrates how to define descriptors for the variables. For this case, the descriptors `u1` through `u7`, `N_x`, and `N_mod` are reported in the sample files to help annotate the data.

7.2.57.6.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)

Export sample sequences enabling file format customization

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

Sample sequences are written on file with a user-defined format. If the keyword `header` is present the heading is used, if the keyword `eval_id` is present the sample number (per level) is included and if `interface_id` is provided then the interface identification is reported.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  output silent
  export_sample_sequence custom_annotated eval_id
```

results in enabling the sample output on file with a customized format including only the progressive number of the sample (`eval_id`).

7.2.57.6.1.1 header

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.57.6.1.2 `eval_id`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.57.6.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.57.6.2 annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [export_sample_sequence](#)
- [annotated](#)

Export sample sequences with descriptors

Specification

Alias: none

Argument(s): none

Default: annotated

Description

Sample sequences are written on file using header descriptors for the data set and a per-level progressive number, `eval_id`. The interface identifier, `interface_id`, is also reported as well as the model descriptors.

7.2.57.6.3 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [export_sample_sequence](#)
- [freeform](#)

Export sample sequences without heading descriptors

Specification

Alias: none

Argument(s): none

Default: annotated

Description

Sample sequences are written on file without using any heading descriptors for the data set. The interface identifier, `interface_id`, and the model descriptors are still reported.

7.2.57.7 allocation_target

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)

Allocation statistics/target for the MLMC sample allocation.

Specification

Alias: none

Argument(s): none

Default: mean

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Allocation target (Group 1)	mean	Fit MLMC sample allocation to control the variance of the estimator for the mean.
			variance	Fit MLMC sample allocation to control the variance of the estimator for the variance.
			standard_deviation	Fit MLMC sample allocation to control the variance of the estimator for the standard deviation.
			scalarization	Fit MLMC sample allocation to a mixture of terms of means and standard deviations.

Description

Computes the variance of the estimator for a given statistics/target and fits sample allocation by solving the corresponding optimization problem. The `allocation_target` specification supports options of `mean` (default), `variance` an analytic approximation is available. Additionally, a numerical optimization can be used in that case, see `variance` option [optimization](#).

Default Behavior "mean"

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = variance
```

uses the variance as sample allocation target by computing its variance.

7.2.57.7.1 mean

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [mean](#)

Fit MLMC sample allocation to control the variance of the estimator for the mean.

Specification

Alias: none

Argument(s): none

Description

Computes the variance of the mean estimator and fits sample allocation by solving the corresponding optimization problem. This optimization problem is solved in analytically closed form.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = mean
```

uses the mean as sample allocation target by computing its variance.

7.2.57.7.2 variance

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [variance](#)

Fit MLMC sample allocation to control the variance of the estimator for the variance.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		optimization	Solve the optimization problem for the sample allocation by numerical optimization in the case of sampling estimator targeting the variance.

Description

Computes the variance of the sampling estimator for the variance and fits sample allocation by solving the corresponding optimization problem. This optimization problem is obtained in closed form with an analytical approximation. Additionally, a numerical optimization can be used in that case, see [optimization](#).

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = variance
```

uses the variance as sample allocation target by computing its variance.

Theory

A single level unbiased estimator for the variance of a generic QoI at the highest level M_L of the hierarchy can be written as

$$\text{Var}[Q_{M_L}] \approx \frac{1}{N_{M_L} - 1} \sum_{i=1}^{N_{M_L}} \left(Q_{M_L}^{(i)} - \mathbb{E}[Q_L] \right)^2.$$

The multilevel extension for this estimator is obtained by writing

$$\text{Var}[Q_L] \approx \hat{Q}_{L,2}^{\text{ML}} = \sum_{\ell=0}^L \hat{Q}_{\ell,2} - \hat{Q}_{\ell-1,2},$$

where

$$\hat{Q}_{\ell,2} = \frac{1}{N_{\ell} - 1} \sum_{i=1}^{N_{\ell}} \left(Q_{\ell}^{(i)} - \hat{Q}_{\ell} \right)^2 \quad \text{and} \quad \hat{Q}_{\ell-1,2} = \frac{1}{N_{\ell-1} - 1} \sum_{i=1}^{N_{\ell-1}} \left(Q_{\ell-1}^{(i)} - \hat{Q}_{\ell-1} \right)^2.$$

As for the expected value case, we want to obtain an optimal sample allocation per level that minimizes the cost to obtain an estimator with a prescribed variance. The variance of the multilevel estimator $\hat{Q}_{L,2}^{\text{ML}}$ can be written as

$$\text{Var} [\hat{Q}_{\ell,2}] \approx \frac{1}{N_\ell} \left(\hat{Q}_{\ell,4} - \frac{N_\ell - 3}{N_\ell - 1} (\hat{Q}_{\ell,2})^2 \right),$$

where $\hat{Q}_{\ell,4}$ denotes the sampling estimator for the fourth order central moment. For more details about the expression that each single term takes in the previous expression, please refer to the Theory Manual.

The final sample allocation is obtained by solving a minimization problem

$$\min_{N_\ell} \sum_{\ell=0}^L \mathcal{C}_\ell N_\ell \quad \text{s.t.} \quad \text{Var} [\hat{Q}_{L,2}^{\text{ML}}] = \varepsilon^2/2.$$

This optimization problem can be solved in two different ways, namely an analytical approximation and by solving a non-linear optimization problem. The analytical approximation follows the approach described in [Pisaroni2017] and introduces a helper variable

$$\hat{V}_{2,\ell} := \text{Var} [\hat{Q}_{\ell,2}] \cdot N_\ell,$$

and the minimization problem is formulated as

$$f(N_\ell, \lambda) = \sum_{\ell=0}^L N_\ell \mathcal{C}_\ell + \lambda \left(\sum_{\ell=0}^L N_\ell^{-1} \hat{V}_{2,\ell} - \varepsilon^2/2 \right).$$

This formulation has a closed form solution (similarly to the expected value case)

$$N_\ell = \frac{2}{\varepsilon^2} \left[\sum_{k=0}^L (\hat{V}_{2,k} \mathcal{C}_k)^{1/2} \right] \sqrt{\frac{\hat{V}_{2,\ell}}{\mathcal{C}_\ell}}.$$

If the option [optimization](#) is specified the previous optimization problem is solved numerically via either OPTPP (default choice) or NPSOL (if available).

See Also

These keywords may also be of interest:

- [optimization](#)

7.2.57.7.2.1 optimization

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [variance](#)
- [optimization](#)

Solve the optimization problem for the sample allocation by numerical optimization in the case of sampling estimator targeting the variance.

Specification

Alias: none

Argument(s): none

Description

Employ numerical optimization to solve the optimization problem for finding the optimal sample allocation when targeting the variance. This option needs NPSOL (preferred choice) or OPTPP (alternative choice) installed.

Default Behavior "off"

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = variance
  optimization
```

uses the variance as sample allocation target and solves the resulting optimization problem using numerical optimization.

7.2.57.7.3 standard_deviation

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [standard_deviation](#)

Fit MLMC sample allocation to control the variance of the estimator for the standard deviation.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		optimization	Solve the optimization problem for the sample allocation by numerical optimization in the case of sampling estimator targeting the standard_deviation.

Description

Computes the variance of the sampling estimator for the standard_deviation and fits sample allocation by solving the corresponding optimization problem.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = standard_deviation
```

uses the `standard_deviation` as sample allocation target by computing its variance.

See Also

These keywords may also be of interest:

- [variance](#)

7.2.57.7.3.1 optimization

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [standard_deviation](#)
- [optimization](#)

Solve the optimization problem for the sample allocation by numerical optimization in the case of sampling estimator targeting the `standard_deviation`.

Specification

Alias: none

Argument(s): none

Description

Employ numerical optimization to solve the optimization problem for finding the optimal sample allocation when targeting the standard deviation. This option needs NPSOL (preferred choice) or OPTPP (alternative choice) installed.

Default Behavior "off"

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = standard_deviation
  optimization
```

uses the standard deviation as sample allocation target and solves the resulting optimization problem using numerical optimization.

7.2.57.7.4 scalarization

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [scalarization](#)

Fit MLMC sample allocation to a mixture of terms of means and standard deviations.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		scalarization_	
	Optional		response_mapping	
	Optional		optimization	Solve the optimization problem for the sample allocation by numerical optimization in the case of sampling estimator targeting the scalarization.

Description

Fit MLMC sample allocation to control the variance of the estimator for a mixture of terms of means and standard deviations. The exact scalarized formulation is given by the keyword `scalarization_response_mapping`.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = scalarization
  scalarization_response_mapping = 1 0 0 0
                                0 0 1 3
```

uses the `standard_deviation` as sample allocation target by computing its variance. In this example, we assume a problem with two responses where the first line in `scalarization_response_mapping` refers to the first response, the second line to the second response. In the first line we only use 1 times the mean as quantity of interest. For the second response, we use 1 time the mean plus 3 times the standard deviation of the second quantity of interest. This behavior mimics the keywords [primary_response_mapping](#) and [secondary_response_mapping](#).

See Also

These keywords may also be of interest:

- [variance](#)

7.2.57.7.4.1 scalarization_response_mapping

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [scalarization](#)
- [scalarization_response_mapping](#)

TODO: Need a Blurb

Specification

Alias: none

Argument(s): REALLIST

Default: default scalarized contribution to mlmc sample allocation (1 Mean, 3 Sigma)

Description

Default Behavior

FAQ

See also [primary_response_mapping](#) and [secondary_response_mapping](#)

7.2.57.7.4.2 optimization

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [allocation_target](#)
- [scalarization](#)
- [optimization](#)

Solve the optimization problem for the sample allocation by numerical optimization in the case of sampling estimator targeting the scalarization.

Specification

Alias: none

Argument(s): none

Description

Employ numerical optimization to solve the optimization problem for finding the optimal sample allocation when targeting the scalarization metric. This option needs NPSOL (preferred choice) or OPTPP (alternative choice) installed.

Default Behavior "off"

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = scalarization
  scalarization_response_mapping = 1 0 0 0
                                0 0 1 3
  optimization
```

uses the scalarization as sample allocation target and solves the resulting optimization problem using numerical optimization.

See Also

These keywords may also be of interest:

- [scalarization](#)

7.2.57.8 qoi_aggregation

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [qoi_aggregation](#)

Aggregation strategy for the Qols statistics for problems with multiple responses in the MLMC algorithm

Specification

Alias: none

Argument(s): none

Default: sum

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Qoi Aggregation (Group 1)	sum	Aggregate the variances over all Qols to generate a target for each level in a MLMC algorithm.
			max	Compute sample allocation for each response and use maximum over responses for each level in a MLMC algorithm

Description

In the multilevel method a variance of the `allocation_target` is computed for each of the responses and their levels $Y_\ell^i, i = 1, \dots, R, \ell = 0, \dots, L$. Setting `qoi_aggregation` describes the rule on how to aggregate those variances over multiple response functions. Supported options are `sum` (default) and `max`. For `sum`, the variances are aggregated and a single sample allocation is computed. For `max`, an individual sample allocation for each response using the respective variances over levels is computed and the maximum over all responses for each level is taken (worst case scenario allocation).

Default Behavior "sum"

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = mean
  qoi_aggregation = sum
```

uses the sum rule to aggregate the variance over the qois.

7.2.57.8.1 sum

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [qoi_aggregation](#)
- [sum](#)

Aggregate the variances over all Qols to generate a target for each level in a MLMC algorithm.

Specification

Alias: none

Argument(s): none

Description

The variances of the `allocation_target` of $Y_\ell^i, i = 1, \dots, R$ are aggregated and a single sample allocation is computed.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = mean
  qoi_aggregation = sum
```

uses the sum rule to aggregate the variance over the qois.

7.2.57.8.2 max

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [qoi_aggregation](#)
- [max](#)

Compute sample allocation for each response and use maximum over responses for each level in a MLMC algorithm

Specification

Alias: none

Argument(s): none

Description

An individual sample allocation for each response using the respective variances of the `allocation_target` of Y_ℓ^i over all levels is computed; the maximum over all responses for each level is taken.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  allocation_target = mean
  qoi_aggregation = max
```

uses the max rule to aggregate the variance over the qois.

7.2.57.9 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [convergence_tolerance](#)

Stopping criterion based on relative error

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

Multilevel sampling seeks an error balance between the estimator variance and the remaining bias error at the highest level, the two contributors to mean squared error (MSE). Since the remaining bias error is generally unknown, the convergence_tolerance is used to provide a error target relative to the Multifidelity Monte Carlo estimator variance resulting from the pilot sample. If the pilot samples are not shaped for the low-fidelity model, i.e. the number of low-fidelity evaluations is equal to the number of high-fidelity evaluations for each level, the Multifidelity estimator falls back to a Multilevel Monte Carlo estimator which is used to assess the estimator pilot samples variance. The samples allocated at each level are proportional to $\frac{1}{\epsilon^2}$, so each order of magnitude reduction in convergence_tolerance will tend to increase the sample allocation by two orders of magnitude. Therefore, this control should be used with care to avoid allocation of huge sample sets that could overrun available memory.

Default Behavior

The default value for convergence_tolerance is currently .0001, which may be too resolved for expensive simulations or high variance QoI.

7.2.57.10 convergence_tolerance_type

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [convergence_tolerance_type](#)

Sets the type of the convergence tolerance. Can be absolute or relative.

Default Behavior "relative"

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Convergence tolerance type (Group 1)	relative	Use a relative converge criterion.
			absolute	Absolute value for the convergence tolerance.

7.2.57.10.1 [relative](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [convergence_tolerance_type](#)
- [relative](#)

Use a relative converge criterion.

Specification

Alias: none

Argument(s): none

Description

Computes an initial tolerance ε_{init}^2 using the pilot samples and applies the value of keyword `convergence_tolerance` c_t to compute the target convergence_tolerance as $\varepsilon_{target}^2 = \varepsilon_{init}^2 \cdot c_t$. Similarly if `convergence_tolerance_target` is equal to `cost_constraint`. See description in current reference manual at `method.multilevel_sampling.convergence_tolerance` as reference as well.

7.2.57.10.2 [absolute](#)

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [convergence_tolerance_type](#)
- [absolute](#)

Absolute value for the convergence tolerance.

Specification

Alias: none

Argument(s): none

Description

Expects an absolute value for the convergence tolerance which will be used as target. If keyword `convergence_tolerance_target` is equal to `variance_constraint`, this value is a target variance (ϵ^2); if keyword `convergence_tolerance_target` is equal to `cost_constraint`, this value is a target cost.

7.2.57.11 `convergence_tolerance_target`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [convergence_tolerance_target](#)

Sets the target of the MLMC sample allocation, i.e. the constraint of the MLMC sample allocation problem

Default Behavior "variance_constraint"

Specification

Alias: none

Argument(s): none

Default: `variance_constraint`

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Convergence tolerance target (Group 1)	Dakota Keyword variance_constraint	Dakota Keyword Description Minimizes the cost while targeting a certain variance given in keyword <code>convergence_tolerance</code> .
			cost_constraint	Minimizes the variance of the estimator while targeting a certain cost given in keyword <code>convergence_tolerance</code> .

7.2.57.11.1 `variance_constraint`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [convergence_tolerance_target](#)
- [variance_constraint](#)

Minimizes the cost while targeting a certain variance given in keyword `convergence_tolerance`.

Specification

Alias: none

Argument(s): none

7.2.57.11.2 cost_constraint

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [convergence_tolerance_target](#)
- [cost_constraint](#)

Minimizes the variance of the estimator while targeting a certain cost given in keyword `convergence_tolerance`.

Specification

Alias: none

Argument(s): none

7.2.57.12 max_iterations

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [max_iterations](#)

Stopping criterion based on number of refinement iterations within the multilevel sample allocation

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

Multilevel sampling is an iterative procedure that estimates the optimal number of samples for each level based on cost and observed variance. On each iteration, additional samples are performed and more accurate variance estimates are computed, leading to updated sample allocations. The process terminates when either no additional samples are allocated or the `max_iterations` control is enforced.

Default Behavior

The default value for `max_iterations` varies by method. For `multilevel_sampling`, the default value is 25.

7.2.57.13 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [max_function_evaluations](#)

Stopping criterion based on maximum function evaluations

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

For multilevel and multifidelity sampling methods, this specification provides a computational budget, indicating that the mean squared error (MSE) of the estimator is to be minimized within this budget. In these methods, the MSE is typically comprised of known estimator variance (stochastic error), neglecting unknown residual bias (deterministic error for the highest-fidelity/most-resolved model).

The budget is provided in terms of the equivalent number of high-fidelity/fine-resolution evaluations. Default value is SZ_MAX (the maximum storable size_t), which is a special value indicating that a budget has not been specified.

7.2.57.14 final_statistics

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)

Indicate the type of final statistics to be returned by a UQ method

Specification

Alias: none

Argument(s): none

Default: qoi_statistics

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final statistics type (Group 1)	estimator_performance	Return estimator performance as the final results of a UQ method
			qoi_statistics	Return the quantity of interest (QoI) statistics as the final results of a UQ method

Description

In nested execution contexts, an inner UQ method may return either a final set of QoI statistics (e.g., for optimization of moments or level mappings over design parameters within optimization under uncertainty) or a set of performance metrics (e.g., for maximization of performance over hyper-parameters within model tuning).

Default Behavior Return a final set of QoI statistics.

7.2.57.14.1 estimator_performance

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [estimator_performance](#)

Return estimator performance as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Description

Instead of QoI moments and level mappings, return the performance of the estimator in terms of its variance and aggregate cost. These metrics are targeted in the context of model tuning for multifidelity UQ methods.

Usage Tips Estimator variance can be minimized when total budget is specified. Alternatively, total budget can be minimized when estimator accuracy is specified. This is managed by the nested model response mappings that define the outer loop objective and constraints, sharing conceptual similarity to the RIA and PMA formulations for optimization with local reliability (refer to RBDO methods in the Theory Manual).

Examples

```
method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
  max_function_evaluations = 1000
```

```

pilot_samples = 25
seed = 8674132
solution_mode pilot_projection
final_statistics estimator_performance

```

7.2.57.14.2 qoi_statistics

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)

Return the quantity of interest (QoI) statistics as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional		distribution	Placeholder for future capabilities

Description

In most nested execution contexts, the outer loop is operating on the set of moments and level mappings returned by an inner loop UQ method. Thus `qoi_statistics` is the default.

Sub-specifications include the type of `final_moments` (none, standard, or central) and the sense of distribution functions (cumulative or complementary).

Examples

```

method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
  max_function_evaluations = 1000
  pilot_samples = 25
  seed = 8674132
  solution_mode online_pilot
  final_statistics qoi_statistics

```

7.2.57.14.2.1 `final_moments`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Final moment type (Group 1)	Dakota Keyword none	Dakota Keyword Description Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to none as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.57.14.2.2 none

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

7.2.57.14.2.3 standard

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.57.14.2.4 central

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)

- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.57.14.2.5 distribution

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)

Placeholder for future capabilities

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required(<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Placeholder for future capabilities
			complementary	Placeholder for future capabilities

Description

At this stage only moments are computed as output in the framework of multilevel sampling techniques. No distribution capabilities are implemented yet.

7.2.57.14.2.6 cumulative

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [cumulative](#)

Placeholder for future capabilities

Specification

Alias: none

Argument(s): none

Description

At this stage only moments are computed as output in the framework of multilevel sampling techniques. No distribution capabilities are implemented yet.

7.2.57.14.2.7 complementary

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [complementary](#)

Placeholder for future capabilities

Specification

Alias: none

Argument(s): none

Description

At this stage only moments are computed as output in the framework of multilevel sampling techniques. No distribution capabilities are implemented yet.

7.2.57.15 rng

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required(<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.57.15.1 mt19937

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.57.15.2 rnum2

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.57.16 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [multilevel_sampling](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.58 multifidelity_sampling

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)

Multifidelity sampling methods for UQ

Specification

Alias: multifidelity_mc mfmc

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		pilot_samples	Initial set of samples for multi-level/multifidelity sampling methods.
	Optional		solution_mode	Solution mode for multi-level/multifidelity methods
	Optional		numerical_solve	Specify the situations where numerical optimization is used for MFMC sample allocation
	Optional		sample_type	Selection of sampling strategy
	Optional		export_sample_sequence	Enable export of multi-level/multifidelity sample sequences to individual files

	Optional	<code>convergence_ - tolerance</code>	Stopping criterion based on relative error reduction
	Optional	<code>max_ iterations</code>	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	<code>max_ function_ - evaluations</code>	Stopping criterion based on maximum function evaluations
	Optional	<code>final_ statistics</code>	Indicate the type of final statistics to be returned by a UQ method
	Optional	<code>rng</code>	Selection of a random number generator
	Optional	<code>model_ pointer</code>	Identifier for model block to be used by a method

Description

An adaptive sampling method that utilizes multifidelity relationships in order to improve efficiency through variance reduction.

Two variants are currently supported, with the former now deprecated and to be replaced by the latter:

- In the case of a hierarchical surrogate model, the two-model approach of Ng and Willcox (2014) is supported and the two most extreme model fidelities or resolutions are employed as the truth model and approximation model.
- In the case of a non-hierarchical surrogate model, the multi-model approach of Peherstorfer et al. (2016) is supported for which all model instances can be integrated into the scheme. Both methods can be used with either a model form sequence or a resolution level sequence.

Control Variate Monte Carlo

In the case of two model fidelities (low fidelity denoted as LF and high fidelity denoted as HF), we employ a control variate approach as described in Ng and Willcox (2014):

$$\hat{Q}_{HF}^{CV} = \hat{Q}_{HF}^{MC} - \beta(\hat{Q}_{LF}^{MC} - \mathbb{E}[Q_{LF}])$$

As opposed to the traditional control variate approach, we do not know $\mathbb{E}[Q_{LF}]$ precisely, but rather we estimate it more accurately than \hat{Q}_{LF}^{MC} based on a sampling increment applied to the LF model. This sampling increment is based again on a total cost minimization procedure that incorporates the relative LF and HF costs and the observed Pearson correlation coefficient ρ_{LH} between Q_{LF} and Q_{HF} . The coefficient β is then determined from the observed LF-HF covariance and LF variance.

Multifidelity Monte Carlo

This approach can be extended to a sequence of low-fidelity approximations using a recursive sampling approach as in Peherstorfer et al. (2016).

$$\hat{Q}_{HF}^{CV} = \hat{Q}_{HF}^{MC} - \sum_{i=1}^M \beta_i (\hat{Q}_{LF_i}^{MC} - \mathbb{E}[Q_{LF_i}])$$

In this case, the variance in the estimate of the i^{th} control mean is reduced by the $(i+1)^{th}$ control variate, such that the variance reduction is limited by the case of an exact estimate of the first control mean (referred to as OCV-1 in Gorodetsky et al., 2020).

Default Behavior

The `multifidelity_sampling` method employs Monte Carlo sample sets by default, but this default can be overridden to use Latin hypercube sample sets using `sample_type lhs`.

Expected Output

The `multifidelity_sampling` method reports estimates of the first four moments and a summary of the evaluations performed for each model fidelity and discretization level. The method does not support any level mappings (response, probability, reliability, generalized reliability) at this time.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Sampling Moments](#) (moments only, not confidence intervals)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

Usage Tips

The `multifidelity_sampling` method can be used in combination with either a hierarchical or non-hierarchical model specification for either a model form sequence or a discretization level sequence. For a model form sequence, each model must provide a scalar `solution_level_cost`. For a discretization level sequence, it is necessary to identify the variable string descriptor that controls the resolution levels using `solution_level_control` as well as the associated array of relative costs using `solution_level_cost`.

The hierarchical two-model approach is a special case of the non-hierarchical multi-model approach. The latter gives identical results to the former when restricted to one approximation model; as such, the hierarchical two-model approach is deprecated.

Examples

We provide an example of a multifidelity Monte Carlo study using a non-hierarchical model specification employing multiple approximations.

The following method block:

```
method,
  model_pointer = 'NONHIER'
  multifidelity_sampling
  pilot_samples = 20 seed = 1237
  max_iterations = 10
  convergence_tolerance = .001
```

specifies MFMC in combination with the model identified by the NONHIER pointer.

This NONHIER model specification provides a one-dimensional sequence, here defined by a single truth model and a set of unordered approximation models, each with a single (or default) discretization level:

```
model,
  id_model = 'NONHIER'
```

```

surrogate non_hierarchical
  truth_model = 'HF'
  unordered_model_fidelities = 'LF1' 'LF2'

model,
  id_model = 'LF1'
  interface_pointer = 'LF1_INT'
  simulation
    solution_level_cost = 0.01

model,
  id_model = 'LF2'
  interface_pointer = 'LF2_INT'
  simulation
    solution_level_cost = 0.1

model,
  id_model = 'HF'
  interface_pointer = 'HF_INT'
  simulation
    solution_level_cost = 1.

```

Refer to `dakota/test/dakota_uq*_cvmc.in` and `dakota/test/dakota_uq*_mfmc.in` in the source distribution for additional examples.

See Also

These keywords may also be of interest:

- [multilevel_sampling](#)
- [multilevel_multifidelity_sampling](#)
- [approximate_control_variate](#)

7.2.58.1 seed_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would

continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For $\$L\$$ levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,  
  model_pointer = 'HIERARCH'  
  multilevel_sampling  
  pilot_samples = 100 50 40 30 20  
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.58.2 `fixed_seed`

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.58.3 pilot_samples

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [pilot_samples](#)

Initial set of samples for multilevel/multifidelity sampling methods.

Specification

Alias: `initial_samples`

Argument(s): INTEGERLIST

Description

The pilot sample provides initial estimates of variance and/or correlation within the first iteration of a multi-level/multifidelity sampling approach. These initial estimates then guide the algorithm toward additional sample increments in order to iteratively achieve a prescribed error at minimum total cost, or alternatively, minimum error for a prescribed budget.

Default Behavior

100 samples per model fidelity and/or discretization level.

Usage Tips

The number of specified values can be none (default values used for all fidelities and levels), one (all fidelities and levels use the same specified value), the number of discretization levels for every model (each model uses the same discretization level profile), or the aggregate number of discretization levels for all models (samples for each discretization level are distinct for each model).

7.2.58.4 solution_mode

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [solution_mode](#)

Solution mode for multilevel/multifidelity methods

Specification

Alias: none

Argument(s): none

Default: online pilot

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Ensemble Sampling Solution Mode (Group 1)	Dakota Keyword online_pilot	Dakota Keyword Description Specify a solution mode that includes the pilot cost within the sample allocation logic
			offline_pilot	Specify a solution mode that excludes the pilot cost from sample allocation logic
			pilot_projection	Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. The pilot sample can be integrated according to one of three solution modes:

- `online_pilot`: include pilot sample cost within sample allocation (default)
- `offline_pilot`: treat pilot cost as offline and exclude from sample allocation
- `pilot_projection`: only perform the pilot sample and only estimate the expected performance based on a projection of the pilot statistics (useful for method selection)

Default Behavior Online pilot

7.2.58.4.1 online_pilot

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [solution_mode](#)
- [online_pilot](#)

Specify a solution mode that includes the pilot cost within the sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as online cost and is included within the optimal sample allocation logic. This mode is the default.

Additional Discussion It is typically advantageous to start from a smaller pilot sample and then rely on the adaptive logic to increase the sample levels to the optimal level required to achieve the desired accuracy or enforce the prescribed budget. Over-estimation of the pilot, while supporting more robust estimation of correlations and variances, can hide the optimal profile and result in a non-optimal allocation of resources.

7.2.58.4.2 offline_pilot

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [solution_mode](#)
- [offline_pilot](#)

Specify a solution mode that excludes the pilot cost from sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as offline cost and is excluded from the optimal sample allocation logic. This mode is typically used for generating reference performance using an "oracle" approach, i.e., specifying a large pilot sample in order to obtain best-case correlation or variance estimates prior to generating a sample profile.

7.2.58.4.3 pilot_projection

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [solution_mode](#)
- [pilot_projection](#)

Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, evaluations are limited to the pilot sample and these initial estimates are used to project the potential performance under the assumption that these estimates are accurate.

This mode is useful for comparing the projected performance of different estimators, which can inform method selection.

7.2.58.5 numerical_solve

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [numerical_solve](#)

Specify the situations where numerical optimization is used for MFMC sample allocation

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Employ numerical solve (Group 1)	fallback	Fall back to a numerical solve when needed for mitigation in MFMC
			override	Replace MFMC analytic allocation with a numerical solution
	Optional (<i>Choose One</i>)	Optimization Solver (Group 2)	sqp	Uses a sequential quadratic programming method for underlying optimization
			nip	Uses a nonlinear interior point method for underlying optimization

Description

Multifidelity Monte Carlo (MFMC) supports an analytic solution for the allocation of samples per model instance based on response correlations and relative simulation cost. In some situations (over-estimated pilot sample, mis-ordered model correlations), this analytic solution may be either sub-optimal or undefined, requiring mitigation.

This specification allows for control of this mitigation; in particular, whether recourse to a numerical solution is strictly a fallback (default) or is desired as an unconditional override (regardless of the need for specific mitigations).

Further, when a numerical solve is employed, it can utilize either the `sqp` or `nip` solver options.

Default Behavior Analytic is preferred, with `fallback` to numerical only when mitigation is required.

7.2.58.5.1 fallback

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [numerical_solve](#)
- [fallback](#)

Fall back to a numerical solve when needed for mitigation in MFMC

Specification

Alias: none

Argument(s): none

Description

Conditionally employ a numerical solve for allocation of samples per model instance within multifidelity Monte Carlo (MFMC). The analytic allocation is employed unless a problem is encountered (mis-ordered model correlations) that requires mitigation.

Examples

```
method,  
  model_pointer = 'ENSEMBLE'  
  multifidelity_sampling  
    numerical_solve nip fallback  
    max_function_evaluations = 1000  
    pilot_samples = 100  
    seed = 8674132
```

7.2.58.5.2 override

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [numerical_solve](#)
- [override](#)

Replace MFMC analytic allocation with a numerical solution

Specification

Alias: none

Argument(s): none

Description

Unconditionally employ a numerical solve for allocation of samples per model instance within multifidelity Monte Carlo (MFMC). The analytic allocation is replaced with numerical solution using either `sqp` or `nip` solvers.

Usage Tips An unconditional override can be appropriate in the case where you select a pilot sample that may be larger than the anticipated optimal sample profile results (pilot over-estimation). In this context, the numerical optimization has greater freedom to work around the incurred pilot cost and find a solution that optimally incorporates the pilot.

Additional Discussion

Examples

```
method,  
  model_pointer = 'ENSEMBLE'  
  multifidelity_sampling  
    numerical_solve nip override  
    max_function_evaluations = 1000  
    pilot_samples = 100  
    seed = 8674132
```

7.2.58.5.3 `sqp`

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [numerical_solve](#)
- [sqp](#)

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `sqp` keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

7.2.58.5.4 `nip`

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [numerical_solve](#)
- [nip](#)

Uses a nonlinear interior point method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

7.2.58.6 `sample_type`

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Sample Type (Group 1)	Dakota Keyword lhs	Dakota Keyword Description Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.58.6.1 `lhs`

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)

- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.58.6.2 random

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.58.7 export_sample_sequence

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [export_sample_sequence](#)

Enable export of multilevel/multifidelity sample sequences to individual files

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

When this option is active, separate output files are written for each unique sample increment and are tagged by algorithm type, simulation interface, iteration count, level count, and the number of samples as described below.

The data content is comprised of the input variables only, without corresponding responses, as an intended use case is to support evaluation of these sample sets offline.

Default Behavior

If not specified, the `annotated` format is assumed.

Expected Output

Separate output files are generated according to the following format: `{ml/cv}_{interface_id}_{iteration_number}_{level_number}_{number_of_samples}.dat`.

With respect to the algorithm type, interface id, and level counter, the following definitions are employed:

- Multilevel MC with hierarchical model: `ml_` is pre-pended for all sample increments and sample sets are tagged with the interface id from the HF model.
- Control Variate MC with hierarchical model (deprecated): `cv_` is prepended for all sample increments, shared sample sets are tagged with the HF interface id, and LF-only refinements are tagged by the LF interface id.
- Multilevel-Control Variate MC with hierarchical model: employs a combination of exports from the two cases above. `ml_` is pre-pended for all HF sample increments and sample sets are tagged with the interface id from the HF model; `cv_` is prepended for all LF increments and sample sets are tagged with the LF interface id. Note that the LF model shares the same `ml_` sample sets, but a redundant file is not created for this data.
- Non-hierarchical approaches (Multifidelity MC, Approximate Control Variate): `cv_` is prepended for all sample increments and sample sets are tagged with the corresponding model interface id. The level number corresponds to the index of the approximation model for LF increments and to the id of the truth model (number of LF approximations + 1) for shared increments.

With respect to iteration count, pilot samples are tagged with iteration 0.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  output silent
  export_sample_sequence
```

results in enabling the sample output of sample increments for each level to individual files using the default `annotated` format.

The following variables block

```
variables,
  id_variables = 'LF_VARS'
  uniform_uncertain = 7
  lower_bounds = 7*-1.
  upper_bounds = 7* 1.
  descriptors 'u1' 'u2' 'u3' 'u4' 'u5' 'u6' 'u7'
  discrete_state_set
  integer = 2
  num_set_values = 4 1
  set_values = 5 15 30 60 # number of spatial coords
               3         # number of Fourier solution modes
  initial_state = 5 3
  descriptors 'N_x' 'N_mod'
```

illustrates how to define descriptors for the variables. For this case, the descriptors `u1` through `u7`, `N_x`, and `N_mod` are reported in the sample files to help annotate the data.

7.2.58.7.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.58.7.1.1 header

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.58.7.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [export_sample_sequence](#)

- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent [custom_annotated](#)

7.2.58.7.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent [custom_annotated](#)

7.2.58.7.2 annotated

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [export_sample_sequence](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.58.7.3 freeform

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [export_sample_sequence](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.58.8 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [convergence_tolerance](#)

Stopping criterion based on relative error reduction

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration through satisfaction of a specified accuracy at minimum total cost.

For multilevel and multifidelity sampling methods, it is a **relative convergence tolerance** on the mean squared error (MSE), where the MSE is typically comprised of known estimator variance (stochastic error), neglecting unknown residual bias (deterministic error).

The reference estimator variance is then defined as follows:

- **Multilevel approaches** (`multilevel_sampling` and `multilevel_multifidelity_sampling`): tolerance relative to initial multilevel Monte Carlo (MLMC) error after pilot sample
- **Control variate approaches** (`multifidelity_sampling` and `approximate_control_variate`): relative to Monte Carlo (MC) error from pilot sample where these are all initial estimator variances for the algorithm, recognizing that control variate approaches can extract no benefit from low-fidelity pilot samples prior to the definition of corresponding sample increments.

Permissible values are between 0 and 1, non-inclusive. Default value is 1.e-4.

7.2.58.9 max_iterations

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.58.10 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [max_function_evaluations](#)

Stopping criterion based on maximum function evaluations

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

For multilevel and multifidelity sampling methods, this specification provides a computational budget, indicating that the mean squared error (MSE) of the estimator is to be minimized within this budget. In these methods, the MSE is typically comprised of known estimator variance (stochastic error), neglecting unknown residual bias (deterministic error for the highest-fidelity/most-resolved model).

The budget is provided in terms of the equivalent number of high-fidelity/fine-resolution evaluations. Default value is SZ_MAX (the maximum storable size_t), which is a special value indicating that a budget has not been specified.

7.2.58.11 final_statistics

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)

Indicate the type of final statistics to be returned by a UQ method

Specification

Alias: none

Argument(s): none

Default: qoi_statistics

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final statistics type (Group 1)	estimator_-performance	Return estimator performance as the final results of a UQ method

			qoi_statistics	Return the quantity of interest (QoI) statistics as the final results of a UQ method
--	--	--	--------------------------------	--

Description

In nested execution contexts, an inner UQ method may return either a final set of QoI statistics (e.g., for optimization of moments or level mappings over design parameters within optimization under uncertainty) or a set of performance metrics (e.g., for maximization of performance over hyper-parameters within model tuning).

Default Behavior Return a final set of QoI statistics.

7.2.58.11.1 estimator_performance

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [estimator_performance](#)

Return estimator performance as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Description

Instead of QoI moments and level mappings, return the performance of the estimator in terms of its variance and aggregate cost. These metrics are targeted in the context of model tuning for multifidelity UQ methods.

Usage Tips Estimator variance can be minimized when total budget is specified. Alternatively, total budget can be minimized when estimator accuracy is specified. This is managed by the nested model response mappings that define the outer loop objective and constraints, sharing conceptual similarity to the RIA and PMA formulations for optimization with local reliability (refer to RBDO methods in the Theory Manual).

Examples

```
method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
    max_function_evaluations = 1000
    pilot_samples = 25
    seed = 8674132
  solution_mode pilot_projection
  final_statistics estimator_performance
```


7.2.58.11.2 qoi_statistics

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)

Return the quantity of interest (QoI) statistics as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional		distribution	Selection of cumulative or complementary cumulative functions

Description

In most nested execution contexts, the outer loop is operating on the set of moments and level mappings returned by an inner loop UQ method. Thus `qoi_statistics` is the default.

Sub-specifications include the type of `final_moments` (none, standard, or central) and the sense of distribution functions (cumulative or complementary).

Examples

```
method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
    max_function_evaluations = 1000
    pilot_samples = 25
    seed = 8674132
    solution_mode online_pilot
    final_statistics qoi_statistics
```

7.2.58.11.2.1 final_moments

- [Keywords Area](#)

- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Select standard rejection
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.58.11.2.2 none

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

7.2.58.11.2.3 standard

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [standard](#)

Select standard rejection

Specification

Alias: none

Argument(s): none

Description

The use of a standard rejection approach in the Monte Carlo Markov Chain algorithm.

7.2.58.11.2.4 central

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.58.11.2.5 distribution

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.58.11.2.6 cumulative

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.58.11.2.7 complementary

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.58.12 rng

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required(<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.58.12.1 mt19937

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.58.12.2 rnum2

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.58.13 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [multifidelity_sampling](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```

    surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.59 multilevel_multifidelity_sampling

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)

Multilevel-Multifidelity sampling methods for UQ

Specification

Alias: multilevel_multifidelity_mc mlmfmc

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets

	Optional	pilot_samples	Initial set of samples for multi-level/multifidelity sampling methods.
	Optional	solution_mode	Solution mode for multi-level/multifidelity methods
	Optional	sample_type	Selection of sampling strategy
	Optional	export_sample_sequence	Enable export of multi-level/multifidelity sample sequences to individual files
	Optional	convergence_tolerance	Stopping criterion based on relative error reduction
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	max_function_evaluations	Stopping criterion based on maximum function evaluations
	Optional	final_statistics	Indicate the type of final statistics to be returned by a UQ method
	Optional	rng	Selection of a random number generator
	Optional	model_pointer	Identifier for model block to be used by a method

Description

An adaptive sampling method that utilizes both multilevel and multifidelity relationships within a hierarchical surrogate model in order to improve efficiency through variance reduction.

In the case of a multilevel relationship, multilevel Monte Carlo methods are used to compute an optimal sample allocation per level, and in the case of a multifidelity relationship, control variate Monte Carlo methods are used to compute an optimal sample allocation per fidelity. These two approaches can also be combined, resulting in the multilevel-multifidelity sampling approach below.

Multilevel Control Variate Monte Carlo

If both multilevel and multifidelity structure are included within a hierarchical model specification, then an inner control variate can be applied across two model fidelities for each level within an outer multilevel approach.

On each level, a control variate is active for the discrepancy Y_ℓ based on

$$Y_\ell^* = Y_\ell + \alpha_\ell (\hat{Y}_\ell^{\text{LF}} - \mathbb{E}[Y_\ell^{\text{LF}}]),$$

where $Y_\ell^{\text{LF}} = \gamma_\ell Q_\ell^{\text{LF}} - Q_\ell^{\text{HF}}$.

The optimal parameter γ_ℓ is computed from the correlation properties between model forms and discretization levels (see the theory manual for further details) and the optimal allocation N_ℓ (per level) is finally obtained from it.

Default Behavior

The `multilevel_multifidelity_sampling` method employs Monte Carlo sample sets by default, but this default can be overridden to use Latin hypercube sample sets using `sample_type lhs`.

Expected Output

The `multilevel_multifidelity_sampling` method reports estimates of the first four moments and a summary of the evaluations performed for each model fidelity and discretization level. The method does not support any level mappings (response, probability, reliability, generalized reliability) at this time.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Sampling Moments](#) (moments only, not confidence intervals)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

Usage Tips

The `multilevel_multifidelity_sampling` method must be used in combination with a hierarchical model specification. The highest and lowest fidelity model must provide multiple discretization levels, for which it is necessary to identify the variable string descriptor that controls the resolution levels using `solution_level_control` as well as the associated array of relative costs using `solution_level_cost`.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_multifidelity_sampling
  pilot_samples = 20 seed = 1237           #s0, #s1, #s2, #s3, #p0, #p1
  convergence_tolerance = .01             #s0, #s2, #s3, #p0, #p1
```

specifies a multilevel-multifidelity Monte Carlo study in combination with the model identified by the HIERARCH pointer. This model specification provides a two-dimensional hierarchy, comprised of two model forms each with four discretization levels:

```
model,
  id_model = 'HIERARCH'
  surrogate hierarchical
  ordered_model_fidelities = 'LF' 'HF'

model,
  id_model = 'LF'
  simulation
  solution_level_control = 'N_x'
  solution_level_cost = 375. 10125. 81000. 648000.

model,
```

```
id_model = 'HF'  
simulation  
  solution_level_control = 'N_x'  
  solution_level_cost = 5.67e+5 4.536e+6 2.1e+7 1.68e+8
```

Refer to `dakota/test/dakota_uq_heat_eq_mlcvmc.in` in the source distribution for this case as well as additional examples.

See Also

These keywords may also be of interest:

- [multilevel_sampling](#)
- [multifidelity_sampling](#)

7.2.59.1 seed_sequence

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For $\$L\$$ levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then

subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,  
  model_pointer = 'HIERARCH'  
  multilevel_sampling  
  pilot_samples = 100 50 40 30 20  
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.59.2 fixed_seed

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.59.3 pilot_samples

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [pilot_samples](#)

Initial set of samples for multilevel/multifidelity sampling methods.

Specification

Alias: initial_samples

Argument(s): INTEGERLIST

Description

The pilot sample provides initial estimates of variance and/or correlation within the first iteration of a multi-level/multifidelity sampling approach. These initial estimates then guide the algorithm toward additional sample increments in order to iteratively achieve a prescribed error at minimum total cost, or alternatively, minimum error for a prescribed budget.

Default Behavior

100 samples per model fidelity and/or discretization level.

Usage Tips

The number of specified values can be none (default values used for all fidelities and levels), one (all fidelities and levels use the same specified value), the number of discretization levels for every model (each model uses the same discretization level profile), or the aggregate number of discretization levels for all models (samples for each discretization level are distinct for each model).

7.2.59.4 solution_mode

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [solution_mode](#)

Solution mode for multilevel/multifidelity methods

Specification

Alias: none

Argument(s): none

Default: online pilot

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Ensemble Sampling Solution Mode (Group 1)	Dakota Keyword online_pilot	Dakota Keyword Description Specify a solution mode that includes the pilot cost within the sample allocation logic
			offline_pilot	Specify a solution mode that excludes the pilot cost from sample allocation logic
			pilot_projection	Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. The pilot sample can be integrated according to one of three solution modes:

- `online_pilot`: include pilot sample cost within sample allocation (default)
- `offline_pilot`: treat pilot cost as offline and exclude from sample allocation
- `pilot_projection`: only perform the pilot sample and only estimate the expected performance based on a projection of the pilot statistics (useful for method selection)

Default Behavior Online pilot

7.2.59.4.1 `online_pilot`

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [solution_mode](#)
- [online_pilot](#)

Specify a solution mode that includes the pilot cost within the sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as online cost and is included within the optimal sample allocation logic. This mode is the default.

Additional Discussion It is typically advantageous to start from a smaller pilot sample and then rely on the adaptive logic to increase the sample levels to the optimal level required to achieve the desired accuracy or enforce the prescribed budget. Over-estimation of the pilot, while supporting more robust estimation of correlations and variances, can hide the optimal profile and result in a non-optimal allocation of resources.

7.2.59.4.2 offline_pilot

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [solution_mode](#)
- [offline_pilot](#)

Specify a solution mode that excludes the pilot cost from sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as offline cost and is excluded from the optimal sample allocation logic. This mode is typically used for generating reference performance using an "oracle" approach, i.e., specifying a large pilot sample in order to obtain best-case correlation or variance estimates prior to generating a sample profile.

7.2.59.4.3 pilot_projection

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [solution_mode](#)
- [pilot_projection](#)

Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, evaluations are limited to the pilot sample and these initial estimates are used to project the potential performance under the assumption that these estimates are accurate.

This mode is useful for comparing the projected performance of different estimators, which can inform method selection.

7.2.59.5 sample_type

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sample Type (Group 1)	lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.59.5.1 lhs

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.59.5.2 random

- [Keywords Area](#)
- [method](#)

- [multilevel_multifidelity_sampling](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.59.6 export_sample_sequence

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [export_sample_sequence](#)

Enable export of multilevel/multifidelity sample sequences to individual files

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (Choose One)	Tabular Format (Group 1)	<code>custom_annotated</code>	Selects custom-annotated tabular file format
			<code>annotated</code>	Selects annotated tabular file format
			<code>freeform</code>	Selects freeform file format

Description

When this option is active, separate output files are written for each unique sample increment and are tagged by algorithm type, simulation interface, iteration count, level count, and the number of samples as described below. The data content is comprised of the input variables only, without corresponding responses, as an intended use case is to support evaluation of these sample sets offline.

Default Behavior

If not specified, the `annotated` format is assumed.

Expected Output

Separate output files are generated according to the following format: `{ml/cv}_{interface_id}_{iteration_number}_{level_number}_{number_of_samples}.dat`.

With respect to the algorithm type, interface id, and level counter, the following definitions are employed:

- Multilevel MC with hierarchical model: `m1_` is pre-pended for all sample increments and sample sets are tagged with the interface id from the HF model.
- Control Variate MC with hierarchical model (deprecated): `cv_` is prepended for all sample increments, shared sample sets are tagged with the HF interface id, and LF-only refinements are tagged by the LF interface id.
- Multilevel-Control Variate MC with hierarchical model: employs a combination of exports from the two cases above. `m1_` is pre-pended for all HF sample increments and sample sets are tagged with the interface id from the HF model; `cv_` is prepended for all LF increments and sample sets are tagged with the LF interface id. Note that the LF model shares the same `m1_` sample sets, but a redundant file is not created for this data.
- Non-hierarchical approaches (Multifidelity MC, Approximate Control Variate): `cv_` is prepended for all sample increments and sample sets are tagged with the corresponding model interface id. The level number corresponds to the index of the approximation model for LF increments and to the id of the truth model (number of LF approximations + 1) for shared increments.

With respect to iteration count, pilot samples are tagged with iteration 0.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  output silent
  export_sample_sequence
```

results in enabling the sample output of sample increments for each level to individual files using the default `annotated` format.

The following variables block

```

variables,
  id_variables = 'LF_VARS'
  uniform_uncertain = 7
  lower_bounds = 7*-1.
  upper_bounds = 7* 1.
  descriptors 'u1' 'u2' 'u3' 'u4' 'u5' 'u6' 'u7'
  discrete_state_set
    integer = 2
    num_set_values = 4 1
    set_values = 5 15 30 60 # number of spatial coords
                  3      # number of Fourier solution modes
    initial_state = 5 3
    descriptors 'N_x' 'N_mod'

```

illustrates how to define descriptors for the variables. For this case, the descriptors `u1` through `u7`, `N_x`, and `N_mod` are reported in the sample files to help annotate the data.

7.2.59.6.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file

	Optional	interface_id	Enable interface ID column in custom-annotated tabular file
--	-----------------	------------------------------	---

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9          1.1          0.0002          0.26           0.76
2             0.90009    1.1 0.0001996404857  0.2601620081    0.759955
3             0.89991    1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.59.6.1.1 header

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.59.6.1.2 `eval_id`

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.59.6.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.59.6.2 annotated

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [export_sample_sequence](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009       1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991       1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.59.6.3 freeform

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [export_sample_sequence](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.59.7 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [convergence_tolerance](#)

Stopping criterion based on relative error reduction

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration through satisfaction of a specified accuracy at minimum total cost.

For multilevel and multifidelity sampling methods, it is a **relative convergence tolerance** on the mean squared error (MSE), where the MSE is typically comprised of known estimator variance (stochastic error), neglecting unknown residual bias (deterministic error).

The reference estimator variance is then defined as follows:

- Multilevel approaches (`multilevel_sampling` and `multilevel_multifidelity_sampling`): tolerance relative to initial multilevel Monte Carlo (MLMC) error after pilot sample
- Control variate approaches (`multifidelity_sampling` and `approximate_control_variate`): relative to Monte Carlo (MC) error from pilot sample where these are all initial estimator variances for the algorithm, recognizing that control variate approaches can extract no benefit from low-fidelity pilot samples prior to the definition of corresponding sample increments.

Permissible values are between 0 and 1, non-inclusive. Default value is 1.e-4.

7.2.59.8 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.59.9 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [max_function_evaluations](#)

Stopping criterion based on maximum function evaluations

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

For multilevel and multifidelity sampling methods, this specification provides a computational budget, indicating that the mean squared error (MSE) of the estimator is to be minimized within this budget. In these methods, the MSE is typically comprised of known estimator variance (stochastic error), neglecting unknown residual bias (deterministic error for the highest-fidelity/most-resolved model).

The budget is provided in terms of the equivalent number of high-fidelity/fine-resolution evaluations. Default value is SZ_MAX (the maximum storable size_t), which is a special value indicating that a budget has not been specified.

7.2.59.10 final_statistics

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)

Indicate the type of final statistics to be returned by a UQ method

Specification

Alias: none

Argument(s): none

Default: qoi_statistics

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final statistics type (Group 1)	estimator_performance	Return estimator performance as the final results of a UQ method
			qoi_statistics	Return the quantity of interest (QoI) statistics as the final results of a UQ method

Description

In nested execution contexts, an inner UQ method may return either a final set of QoI statistics (e.g., for optimization of moments or level mappings over design parameters within optimization under uncertainty) or a set of performance metrics (e.g., for maximization of performance over hyper-parameters within model tuning).

Default Behavior Return a final set of QoI statistics.

7.2.59.10.1 estimator_performance

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [estimator_performance](#)

Return estimator performance as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Description

Instead of QoI moments and level mappings, return the performance of the estimator in terms of its variance and aggregate cost. These metrics are targeted in the context of model tuning for multifidelity UQ methods.

Usage Tips Estimator variance can be minimized when total budget is specified. Alternatively, total budget can be minimized when estimator accuracy is specified. This is managed by the nested model response mappings that define the outer loop objective and constraints, sharing conceptual similarity to the RIA and PMA formulations for optimization with local reliability (refer to RBDO methods in the Theory Manual).

Examples

```
method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
  max_function_evaluations = 1000
```

```

pilot_samples = 25
seed = 8674132
solution_mode pilot_projection
final_statistics estimator_performance

```

7.2.59.10.2 qoi_statistics

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)

Return the quantity of interest (QoI) statistics as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional		distribution	Selection of cumulative or complementary cumulative functions

Description

In most nested execution contexts, the outer loop is operating on the set of moments and level mappings returned by an inner loop UQ method. Thus `qoi_statistics` is the default.

Sub-specifications include the type of `final_moments` (none, standard, or central) and the sense of distribution functions (cumulative or complementary).

Examples

```

method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
  max_function_evaluations = 1000
  pilot_samples = 25
  seed = 8674132
  solution_mode online_pilot
  final_statistics qoi_statistics

```


7.2.59.10.2.1 final_moments

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Select standard rejection
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.59.10.2.2 none

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

7.2.59.10.2.3 standard

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [standard](#)

Select standard rejection

Specification

Alias: none

Argument(s): none

Description

The use of a standard rejection approach in the Monte Carlo Markov Chain algorithm.

7.2.59.10.2.4 central

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.59.10.2.5 distribution

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.59.10.2.6 cumulative

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.59.10.2.7 complementary

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.59.11 rng

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.59.11.1 mt19937

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.59.11.2 rnum2

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```


7.2.59.12 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [multilevel_multifidelity_sampling](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```

surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system async evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.60 approximate_control_variate

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)

Approximate control variate (ACV) sampling methods for UQ

Specification

Alias: acv_sampling

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group Solution Approach (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		acv_independent_sampling	Approximate control variate (ACV) algorithm that employs independent samples per model

			acv_multifidelity	Approximate control variate (ACV) algorithm that mimics MFMC by employing a nested pyramid sample pattern. This ACV variant uses sample set definitions that are similar to multifidelity Monte Carlo (MFMC), in that sample sets are nested with each new level adding an increment on top of the previous.
			acv_kl	Approximate control variate (ACV) algorithm that optimizes a mix of hierarchical and non-hierarchical schemes.
	Optional		pilot_samples	Initial set of samples for multi-level/multifidelity sampling methods.
	Optional		solution_mode	Solution mode for multi-level/multifidelity methods.
	Optional		truth_fixed_by_pilot	Option to suppress any increment to the number of truth samples.
	Optional (<i>Choose One</i>)	Optimization Solver (Group 2)	sqp	Uses a sequential quadratic programming method for underlying optimization.

		nip	Uses a nonlinear interior point method for underlying optimization
	Optional	seed_sequence	Sequence of seed values for multi-stage random sampling
	Optional	fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional	sample_type	Selection of sampling strategy
	Optional	export_sample_sequence	Enable export of multi-level/multifidelity sample sequences to individual files
	Optional	convergence_tolerance	Stopping criterion based on relative error reduction
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	max_function_evaluations	Stopping criterion based on maximum function evaluations
	Optional	final_statistics	Indicate the type of final statistics to be returned by a UQ method
	Optional	rng	Selection of a random number generator

	Optional	<code>model_pointer</code>	Identifier for model block to be used by a method
--	-----------------	----------------------------	---

Description

An adaptive sampling method that utilizes multifidelity relationships in order to improve efficiency through variance reduction. It employs a non-hierarchical model to manage an unordered set of lower-fidelity approximations to a single truth model.

Compared to multifidelity Monte Carlo (MFMC), ACV relaxes the nested sampling of a recursive emulator, instead targeting the truth model's variance with each control variate pair. While the ensemble of control variates appears identical to MFMC:

$$\hat{Q}_{HF}^{CV} = \hat{Q}_{HF}^{MC} - \sum_{i=1}^M \beta_i (\hat{Q}_{LF_i}^{MC} - \mathbb{E}[Q_{LF_i}])$$

the sample patterns used for the constituent estimators differ as depicted in Gorodetsky et al. (2020), Figure 2. Two ACV variants are currently implemented, ACV-MF and ACV-IS, with ACV-KL to follow.

Default Behavior

The `approximate_control_variate` method employs Monte Carlo sample sets by default, but this default can be overridden to use Latin hypercube sample sets using `sample_type lhs`.

Expected Output

The `approximate_control_variate` method reports estimates of the first four moments and a summary of the evaluations performed for each model fidelity and discretization level. The method does not support any level mappings (response, probability, reliability, generalized reliability) at this time.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Sampling Moments](#) (moments only, not confidence intervals)

In addition, the execution group has the attribute `equiv_hf_evals`, which records the equivalent number of high-fidelity evaluations.

Usage Tips

The `approximate_control_variate` method must be used in combination with a non-hierarchical model specification that defines either a model form sequence or a discretization level sequence. For a model form sequence, each model must provide a scalar `solution_level_cost`. For a discretization level sequence, `solution_level_control` must identify the variable string descriptor that controls the resolution levels and the associated array of relative costs must be provided using `solution_level_cost`.

Examples

The following method block:

```
method,
  model_pointer = 'NONHIER'
  approximate_control_variate
    acv_mf nip
    pilot_samples = 20 seed = 1237
    max_iterations = 10
    convergence_tolerance = .001
```

specifies ACV-MF using the nonlinear interior point (NIP) solver in combination with the model identified by the NONHIER pointer.

This NONHIER model specification provides a one-dimensional sequence, here defined by a single truth model and a set of unordered approximation models, each with a single (or default) discretization level:

```

model,
  id_model = 'NONHIER'
  surrogate non_hierarchical
  truth_model = 'HF'
  unordered_model_fidelities = 'LF1' 'LF2'

model,
  id_model = 'LF1'
  interface_pointer = 'LF1_INT'
  simulation
  solution_level_cost = 1

model,
  id_model = 'LF2'
  interface_pointer = 'LF2_INT'
  simulation
  solution_level_cost = 16

model,
  id_model = 'HF'
  interface_pointer = 'HF_INT'
  simulation
  solution_level_cost = 256.

```

Refer to `dakota/test/dakota_uq_diffusion_acv3_cost4.in` and `dakota/test/dakota_uq-_tunable_acv.in` in the source distribution for this case as well as additional examples.

Refer to [Gorodetsky et al., JCP (408), 2020] for more detailed algorithm descriptions, theoretical considerations, and a helpful sample set diagram.

See Also

These keywords may also be of interest:

- [multifidelity_sampling](#)

7.2.60.1 acv_independent_sampling

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [acv_independent_sampling](#)

Approximate control variate (ACV) algorithm that employs independent samples per model

Specification

Alias: `acv_js`

Argument(s): none

Description

This ACV variant uses sample set definitions that are similar to multilevel Monte Carlo (MLMC), in that independent sample sets are defined for each level and span a pair of consecutive models within a sequence.

Theory

Refer to [37] for a helpful sample set diagram.

7.2.60.2 acv_multifidelity

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [acv_multifidelity](#)

Approximate control variate (ACV) algorithm that mimics MFMC by employing a nested pyramid sample pattern

This ACV variant uses sample set definitions that are similar to multifidelity Monte Carlo (MFMC), in that sample sets are nested with each new level adding an increment on top of the previous.

Specification

Alias: acv_mf

Argument(s): none

Theory

Refer to [37] for a helpful sample set diagram.

7.2.60.3 acv_kl

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [acv_kl](#)

Approximate control variate (ACV) algorithm that optimizes a mix of hierarchial and non-hierarchial schemes

Specification

Alias: acv_adaptive

Argument(s): none

Description

This ACV variant is not yet implemented. ACV-KL enumerates a set of free parameters, K and L, that mix hierarchial and non-hierarchial sampling schemes in order to obtain the best estimator for a particular budget or accuracy target.

Theory

Refer to [37] for additional details.

7.2.60.4 pilot_samples

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [pilot_samples](#)

Initial set of samples for multilevel/multifidelity sampling methods.

Specification

Alias: initial_samples

Argument(s): INTEGERLIST

Description

The pilot sample provides initial estimates of variance and/or correlation within the first iteration of a multi-level/multifidelity sampling approach. These initial estimates then guide the algorithm toward additional sample increments in order to iteratively achieve a prescribed error at minimum total cost, or alternatively, minimum error for a prescribed budget.

Default Behavior

100 samples per model fidelity and/or discretization level.

Usage Tips

The number of specified values can be none (default values used for all fidelities and levels), one (all fidelities and levels use the same specified value), the number of discretization levels for every model (each model uses the same discretization level profile), or the aggregate number of discretization levels for all models (samples for each discretization level are distinct for each model).

7.2.60.5 solution_mode

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [solution_mode](#)

Solution mode for multilevel/multifidelity methods

Specification

Alias: none

Argument(s): none

Default: online pilot

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Ensemble Sampling Solution Mode (Group 1)	online_pilot	Specify a solution mode that includes the pilot cost within the sample allocation logic
			offline_pilot	Specify a solution mode that excludes the pilot cost from sample allocation logic
			pilot_projection	Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. The pilot sample can be integrated according to one of three solution modes:

- `online_pilot`: include pilot sample cost within sample allocation (default)
- `offline_pilot`: treat pilot cost as offline and exclude from sample allocation
- `pilot_projection`: only perform the pilot sample and only estimate the expected performance based on a projection of the pilot statistics (useful for method selection)

Default Behavior Online pilot

7.2.60.5.1 `online_pilot`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [solution_mode](#)
- [online_pilot](#)

Specify a solution mode that includes the pilot cost within the sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as online cost and is included within the optimal sample allocation logic. This mode is the default.

Additional Discussion It is typically advantageous to start from a smaller pilot sample and then rely on the adaptive logic to increase the sample levels to the optimal level required to achieve the desired accuracy or enforce the prescribed budget. Over-estimation of the pilot, while supporting more robust estimation of correlations and variances, can hide the optimal profile and result in a non-optimal allocation of resources.

7.2.60.5.2 `offline_pilot`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [solution_mode](#)
- [offline_pilot](#)

Specify a solution mode that excludes the pilot cost from sample allocation logic

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, the cost of pilot sampling is treated as offline cost and is excluded from the optimal sample allocation logic. This mode is typically used for generating reference performance using an "oracle" approach, i.e., specifying a large pilot sample in order to obtain best-case correlation or variance estimates prior to generating a sample profile.

7.2.60.5.3 `pilot_projection`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [solution_mode](#)
- [pilot_projection](#)

Specify a solution mode that estimates performance based on projecting initial correlation/variance estimates from a pilot sample

Specification

Alias: none

Argument(s): none

Description

Multilevel / multifidelity sampling methods are adaptive UQ methods that utilize a pilot sample to estimate an initial set of correlations or variances, and then augment the pilot with additional sample increments to optimally allocate resources. In this mode, evaluations are limited to the pilot sample and these initial estimates are used to project the potential performance under the assumption that these estimates are accurate.

This mode is useful for comparing the projected performance of different estimators, which can inform method selection.

7.2.60.6 `truth_fixed_by_pilot`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [truth_fixed_by_pilot](#)

Option to suppress any increment to the number of truth samples

Specification

Alias: none

Argument(s): none

Description

In some cases, it may be impractical to increment an initial set of truth model samples, for example due to their extreme computational cost or due to the use of legacy data sets that can no longer be augmented. To fix the number of truth model evaluations to only those prescribed in the pilot sample, the `truth_fixed_by_pilot` keyword can be used.

Default Behavior

The default is to allow the number of truth samples to advance beyond those specified in the pilot, while seeking an optimal sample profile.

Additional Discussion

The optimization problem is formulated using linear constraints and a reduced set of design variables, one per approximation model.

This option is most relevant in the default `online_pilot` solution mode, which would normally iterate to determine the optimal number of truth samples. With `truth_fixed_by_pilot`, the number of truth samples is fixed within the optimization and no iteration occurs.

For the `pilot_projection` solution mode, the optimization formulation searches only over the approximation samples, providing an optimal projection subject to the fixed truth constraint.

The `offline_pilot` solution mode is not iterative and assumes that the offline pilot has accurately estimated the correlations and variances. In this case, the online profile is independent and it is not necessary to refine the truth model samples beyond the initial estimate of the optimal allocation. The `truth_fixed_by_pilot` specification has no additional effect beyond these existing constraints.

7.2.60.7 `sqp`

- [Keywords Area](#)

- [method](#)
- [approximate_control_variate](#)
- [sqp](#)

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `sqp` keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

7.2.60.8 nip

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [nip](#)

Uses a nonlinear interior point method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

7.2.60.9 seed_sequence

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [seed_sequence](#)

Sequence of seed values for multi-stage random sampling

Specification

Alias: none

Argument(s): INTEGERLIST

Default: system-generated (non-repeatable)

Description

This specification allows the stipulation of seed values (passed to a random number generator) for each of a set of stages within a multi-stage random sampling algorithm.

This is particularly useful for reusing sample sets (via restart) that were generated in some other context (e.g., from a single-stage study) within a multi-stage algorithm execution. Normally the random number sequence would continue from a seed specified for the initial sample set, making subsequent sample sets more difficult to recreate outside of their original context. With finer granularity in the seed specification for multi-stage samplers, intermediate portions of a sampling sequence can now be more readily recreated and reused.

Default Behavior

As for the scalar case, the default is no seed control, such that a query to a system clock will be used to randomize results.

Usage Tips

For L levels in a multilevel method, the `seed_sequence` can be of any length and seeds will be assigned up to the number of values that are provided or up to the number of levels in the sequence. Thus, a `seed_sequence` of length 1 recovers the behavior of a scalar seed.

When `fixed_seed` is unspecified, the `seed_sequence` specification does not extend to the refinement of the sample set. For example, the value from the `seed_sequence` defines the initial sample set for a level, but then subsequent sample sets generated as data increments will allow the random number sequence to continue without resetting the seed value. This reflects the need to avoid reusing previous sample values when generating sample augmentations.

Examples

Defining a multilevel Monte Carlo in the following way

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 100 50 40 30 20
  seed_sequence = 1234 2345 3456 4567 5678
```

allows the reuse (via restart) of intermediate sample sets from other sources by matching the seed specifications for the targeted model resolutions.

See Also

These keywords may also be of interest:

- [seed](#)
- [fixed_seed](#)

7.2.60.10 fixed_seed

- [Keywords Area](#)

- [method](#)
- [approximate_control_variate](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.60.11 sample_type

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sample Type (Group 1)	lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.2.60.11.1 lhs

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.60.11.2 random

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```


7.2.60.12 export_sample_sequence

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [export_sample_sequence](#)

Enable export of multilevel/multifidelity sample sequences to individual files

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

When this option is active, separate output files are written for each unique sample increment and are tagged by algorithm type, simulation interface, iteration count, level count, and the number of samples as described below. The data content is comprised of the input variables only, without corresponding responses, as an intended use case is to support evaluation of these sample sets offline.

Default Behavior

If not specified, the `annotated` format is assumed.

Expected Output

Separate output files are generated according to the following format: `{ml/cv}_{interface_id}_{iteration_number}_{level_number}_{number_of_samples}.dat`.

With respect to the algorithm type, interface id, and level counter, the following definitions are employed:

- Multilevel MC with hierarchical model: `ml_` is pre-pended for all sample increments and sample sets are tagged with the interface id from the HF model.
- Control Variate MC with hierarchical model (deprecated): `cv_` is prepended for all sample increments, shared sample sets are tagged with the HF interface id, and LF-only refinements are tagged by the LF interface id.
- Multilevel-Control Variate MC with hierarchical model: employs a combination of exports from the two cases above. `ml_` is pre-pended for all HF sample increments and sample sets are tagged with the interface id from the HF model; `cv_` is prepended for all LF increments and sample sets are tagged with the LF interface id. Note that the LF model shares the same `ml_` sample sets, but a redundant file is not created for this data.

- Non-hierarchical approaches (Multifidelity MC, Approximate Control Variate): `cv_` is prepended for all sample increments and sample sets are tagged with the corresponding model interface id. The level number corresponds to the index of the approximation model for LF increments and to the id of the truth model (number of LF approximations + 1) for shared increments.

With respect to iteration count, pilot samples are tagged with iteration 0.

Examples

The following method block

```
method,
  model_pointer = 'HIERARCH'
  multilevel_sampling
  pilot_samples = 20 seed = 1237
  convergence_tolerance = .01
  output silent
  export_sample_sequence
```

results in enabling the sample output of sample increments for each level to individual files using the default annotated format.

The following variables block

```
variables,
  id_variables = 'LF_VARS'
  uniform_uncertain = 7
  lower_bounds = 7*-1.
  upper_bounds = 7* 1.
  descriptors 'u1' 'u2' 'u3' 'u4' 'u5' 'u6' 'u7'
  discrete_state_set
  integer = 2
  num_set_values = 4 1
  set_values = 5 15 30 60 # number of spatial coords
                3      # number of Fourier solution modes
  initial_state = 5 3
  descriptors 'N_x' 'N_mod'
```

illustrates how to define descriptors for the variables. For this case, the descriptors `u1` through `u7`, `N_x`, and `N_mod` are reported in the sample files to help annotate the data.

7.2.60.12.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [export_sample_sequence](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

%eval_id	x1	x2	obj_fn	nl_n_ineq_con_1	nl_n_ineq_con_2
1	0.9	1.1	0.0002	0.26	0.76
2	0.90009	1.1	0.0001996404857	0.2601620081	0.759955
3	0.89991	1.1	0.0002003604863	0.2598380081	0.760045
...					

7.2.60.12.1.1 header

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.60.12.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.60.12.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [export_sample_sequence](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.60.12.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [export_sample_sequence](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.60.12.3 freeform

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [export_sample_sequence](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.60.13 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [convergence_tolerance](#)

Stopping criterion based on relative error reduction

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration through satisfaction of a specified accuracy at minimum total cost.

For multilevel and multifidelity sampling methods, it is a **relative convergence tolerance** on the mean squared error (MSE), where the MSE is typically comprised of known estimator variance (stochastic error), neglecting unknown residual bias (deterministic error).

The reference estimator variance is then defined as follows:

- Multilevel approaches (`multilevel_sampling` and `multilevel_multifidelity_sampling`): tolerance relative to initial multilevel Monte Carlo (MLMC) error after pilot sample
- Control variate approaches (`multifidelity_sampling` and `approximate_control_variate`): relative to Monte Carlo (MC) error from pilot sample where these are all initial estimator variances for the algorithm, recognizing that control variate approaches can extract no benefit from low-fidelity pilot samples prior to the definition of corresponding sample increments.

Permissible values are between 0 and 1, non-inclusive. Default value is 1.e-4.

7.2.60.14 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.60.15 max_function_evaluations

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [max_function_evaluations](#)

Stopping criterion based on maximum function evaluations

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

For multilevel and multifidelity sampling methods, this specification provides a computational budget, indicating that the mean squared error (MSE) of the estimator is to be minimized within this budget. In these methods, the MSE is

typically comprised of known estimator variance (stochastic error), neglecting unknown residual bias (deterministic error for the highest-fidelity/most-resolved model).

The budget is provided in terms of the equivalent number of high-fidelity/fine-resolution evaluations. Default value is SZ_MAX (the maximum storable size_t), which is a special value indicating that a budget has not been specified.

7.2.60.16 final_statistics

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)

Indicate the type of final statistics to be returned by a UQ method

Specification

Alias: none

Argument(s): none

Default: qoi_statistics

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Final statistics type (Group 1)	Dakota Keyword estimator_performance	Dakota Keyword Description Return estimator performance as the final results of a UQ method
			qoi_statistics	Return the quantity of interest (QoI) statistics as the final results of a UQ method

Description

In nested execution contexts, an inner UQ method may return either a final set of QoI statistics (e.g., for optimization of moments or level mappings over design parameters within optimization under uncertainty) or a set of performance metrics (e.g., for maximization of performance over hyper-parameters within model tuning).

Default Behavior Return a final set of QoI statistics.

7.2.60.16.1 estimator_performance

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [estimator_performance](#)

Return estimator performance as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Description

Instead of QoI moments and level mappings, return the performance of the estimator in terms of its variance and aggregate cost. These metrics are targeted in the context of model tuning for multifidelity UQ methods.

Usage Tips Estimator variance can be minimized when total budget is specified. Alternatively, total budget can be minimized when estimator accuracy is specified. This is managed by the nested model response mappings that define the outer loop objective and constraints, sharing conceptual similarity to the RIA and PMA formulations for optimization with local reliability (refer to RBDO methods in the Theory Manual).

Examples

```
method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
    max_function_evaluations = 1000
    pilot_samples = 25
    seed = 8674132
    solution_mode pilot_projection
    final_statistics estimator_performance
```

7.2.60.16.2 qoi_statistics

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [qoi_statistics](#)

Return the quantity of interest (QoI) statistics as the final results of a UQ method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		final_moments	Output moments of the specified type and include them within the set of final statistics.

	Optional	distribution	Selection of cumulative or complementary cumulative functions
--	-----------------	------------------------------	---

Description

In most nested execution contexts, the outer loop is operating on the set of moments and level mappings returned by an inner loop UQ method. Thus `qoi_statistics` is the default.

Sub-specifications include the type of `final_moments` (`none`, `standard`, or `central`) and the sense of distribution functions (`cumulative` or `complementary`).

Examples

```
method,
  id_method = 'UQ'
  model_pointer = 'ENSEMBLE'
  multifidelity_sampling
    max_function_evaluations = 1000
    pilot_samples = 25
    seed = 8674132
  solution_mode online_pilot
  final_statistics qoi_statistics
```

7.2.60.16.2.1 final_moments

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.

			standard	Select standard rejection
			central	Output central moments and include them within the set of final statistics.

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.60.16.2.2 none

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)

- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,  
  sampling  
    samples = 50 seed = 1234  
    response_levels = 3.6e+11 1.2e+05 3.5e+05  
    final_moments none
```

7.2.60.16.2.3 standard

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [standard](#)

Select standard rejection

Specification

Alias: none

Argument(s): none

Description

The use of a standard rejection approach in the Monte Carlo Markov Chain algorithm.

7.2.60.16.2.4 central

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,  
  sampling  
    samples = 50 seed = 1234  
    response_levels = 3.6e+11 1.2e+05 3.5e+05  
    final_moments central
```

7.2.60.16.2.5 distribution

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.60.16.2.6 cumulative

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.60.16.2.7 complementary

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [final_statistics](#)
- [qoi_statistics](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.60.17 rng

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.60.17.1 mt19937

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.60.17.2 rnum2

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.60.18 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [approximate_control_variate](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.61 importance_sampling

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)

Importance sampling

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)
- [aleatory_uncertainty_quantification_methods](#)
- [sampling](#)

Specification

Alias: nond_importance_sampling

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		samples	Number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Required (<i>Choose One</i>)	Importance Sampling Approach (Group 1)	import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods

	Optional	convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	gen_reliability_ - levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions
	Optional	rng	Selection of a random number generator
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The `importance_sampling` method is based on ideas in reliability modeling.

An initial Latin Hypercube sampling is performed to generate an initial set of samples. These initial samples are augmented with samples from an importance density as follows:

- The variables are transformed to standard normal space.
- In the transformed space, the importance density is a set of normal densities centered around points which are in the failure region.
- Note that this is similar in spirit to the reliability methods, in which importance sampling is centered around a Most Probable Point (MPP).
- In the case of the LHS samples, the importance sampling density will simply be a mixture of normal distributions centered around points in the failure region.

Options

Choose one of the importance sampling options:

- `import`

- `adapt_import`
- `mm_adapt_import`

The options for importance sampling are as follows: `import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). `adapt_import` is the same as `import` but is performed iteratively until the failure probability estimate converges. `mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

Theory

Importance sampling is a method that allows one to estimate statistical quantities such as failure probabilities (e.g. the probability that a response quantity will exceed a threshold or fall below a threshold value) in a way that is more efficient than Monte Carlo sampling. The core idea in importance sampling is that one generates samples that preferentially samples important regions in the space (e.g. in or near the failure region or user-defined region of interest), and then appropriately weights the samples to obtain an unbiased estimate of the failure probability [79]. In importance sampling, the samples are generated from a density which is called the importance density: it is not the original probability density of the input distributions. The importance density should be centered near the failure region of interest. For black-box simulations such as those commonly interfaced with Dakota, it is difficult to specify the importance density a priori: the user often does not know where the failure region lies, especially in a high-dimensional space. [81]. We have developed two importance sampling approaches which do not rely on the user explicitly specifying an importance density.

See Also

These keywords may also be of interest:

- [adaptive_sampling](#)
- [gpais](#)
- [local_reliability](#)
- [global_reliability](#)
- [sampling](#)
- [polynomial_chaos](#)
- [stoch_collocation](#)

7.2.61.1 samples

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: initial_samples

Argument(s): INTEGER

Default: 0

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10*\text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N*(\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.61.2 seed

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.61.3 import

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.61.4 adapt_import

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.61.5 `mm_adapt_import`

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.61.6 `refinement_samples`

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.61.7 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.61.8 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.61.9 response_levels

- [Keywords Area](#)
- [method](#)

- [importance_sampling](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05

```

7.0000000000e+04  7.8702465755e+04  1.0341896485e-05
PDF for response_fn_3:
  Bin Lower          Bin Upper          Density Value
  -----
2.3796737090e+05  3.5000000000e+05  4.2844660868e-06
3.5000000000e+05  4.0000000000e+05  8.6000000000e-06
4.0000000000e+05  4.5000000000e+05  1.8000000000e-06

Level mappings for each response function:
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
3.6000000000e+11  5.5000000000e-01
4.0000000000e+11  3.8000000000e-01
4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
6.0000000000e+04  6.1000000000e-01
6.5000000000e+04  2.9000000000e-01
7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
3.5000000000e+05  5.2000000000e-01
4.0000000000e+05  9.0000000000e-02
4.5000000000e+05  0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.61.9.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities,

both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.61.9.2 compute

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.61.9.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.61.9.2.2 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.61.9.2.3 system

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.61.9.2.4 series

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.61.9.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.61.10 probability_levels

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Optional		num_probability_levels	Specify which probability_levels correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
```

```

uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors        = 'TF1u' 'TF2u'
weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors        = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
  num_pairs         = 3      4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors        = 'TF1h' 'TF2h'
histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors        = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		

3.6997214153e+05	3.0000000000e-01
3.8100966235e+05	2.0000000000e-01
4.4111498127e+05	0.0000000000e+00

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.61.10.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page

7.2.61.11 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.61.11.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.61.12 distribution

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
```

```
sample_type lhs
samples = 10
distribution cumulative
```

7.2.61.12.1 cumulative

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
  sample_type lhs
  samples = 10
  distribution cumulative
```

7.2.61.12.2 complementary

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.61.13 rng

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister

			<code>rnum2</code>	Generates pseudo-random numbers using the Pecos package
--	--	--	--------------------	---

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.61.13.1 mt19937

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.61.13.2 rnum2

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.61.14 model_pointer

- [Keywords Area](#)
- [method](#)
- [importance_sampling](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.62 gpais

- [Keywords Area](#)
- [method](#)
- [gpais](#)

Gaussian Process Adaptive Importance Sampling

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)

Specification

Alias: gaussian_process_adaptive_importance_sampling

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		build_samples	Number of initial model evaluations used in build phase
	Optional		seed	Seed of the random number generator
	Optional		samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)

	Optional	import_build_points_file	File containing points you wish to use to build a surrogate
	Optional	export_approx_points_file	Output file for surrogate model value evaluations
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions
	Optional	rng	Selection of a random number generator
	Optional	model_pointer	Identifier for model block to be used by a method

Description

`gpais` is recommended for problems that have a relatively small number of input variables (e.g. less than 10-20). This method, Gaussian Process Adaptive Importance Sampling, is outlined in the paper [?].

This method starts with an initial set of LHS samples and adds samples one at a time, with the goal of adaptively improving the estimate of the ideal importance density during the process. The approach uses a mixture of component densities. An iterative process is used to construct the sequence of improving component densities. At each iteration, a Gaussian process (GP) surrogate is used to help identify areas in the space where failure is likely to occur. The GPs are not used to directly calculate the failure probability; they are only used to approximate the importance density. Thus, the Gaussian process adaptive importance sampling algorithm overcomes limitations involving using a potentially inaccurate surrogate model directly in importance sampling calculations.

See Also

These keywords may also be of interest:

- [adaptive_sampling](#)
- [local_reliability](#)
- [global_reliability](#)
- [sampling](#)
- [importance_sampling](#)
- [polynomial_chaos](#)
- [stoch_collocation](#)

7.2.62.1 build_samples

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [build_samples](#)

Number of initial model evaluations used in build phase

Specification

Alias: samples

Argument(s): INTEGER

Description

The number of build points or training points used in the initial phase of an algorithm or model construction. Typically these are the initial set of data points used to construct (train) a surrogate model (emulator). If the number of `build_samples` is less than the minimum number of points required to build the surrogate, Dakota will augment the samples to obtain the minimum required.

Examples

Perform GP-based adaptive importance sampling, building the GP with 100 points and then performing 100 approximate evaluations to evaluate the probability.

```
method
  gpais
    build_samples = 100
    samples_on_emulator = 100
    max_iterations = 5
    response_levels = -1.065
```

7.2.62.2 seed

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.62.3 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: none

Argument(s): INTEGER

Default: 10000

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.62.4 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

		freeform	Selects freeform file format
	Optional	active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.62.4.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```

%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9        1.1         0.0002         0.26           0.76
2             0.90009    1.1 0.0001996404857  0.2601620081   0.759955
3             0.89991    1.1 0.0002003604863  0.2598380081   0.760045
...

```

7.2.62.4.1.1 header

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.62.4.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.62.4.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.62.4.2 annotated

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.62.4.3 freeform

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.62.4.4 active_only

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.62.5 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
--	--	--	--------------------------	------------------------------

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.62.5.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) annotated.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.62.5.1.1 header

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [export_approx_points_file](#)

- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent [custom_annotated](#)

7.2.62.5.1.2 [eval_id](#)

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no [eval_id](#) column

Description

See description of parent [custom_annotated](#)

7.2.62.5.1.3 [interface_id](#)

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.62.5.2 annotated

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.62.5.3 freeform

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.62.6 max_iterations

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.62.7 response_levels

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'
```

```

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06
4.0000000000e+05	4.5000000000e+05	1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
3.6000000000e+11	5.5000000000e-01		
4.0000000000e+11	3.8000000000e-01		
4.4000000000e+11	2.3000000000e-01		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
6.0000000000e+04	6.1000000000e-01		
6.5000000000e+04	2.9000000000e-01		
7.0000000000e+04	9.0000000000e-02		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
3.5000000000e+05	5.2000000000e-01		
4.0000000000e+05	9.0000000000e-02		
4.5000000000e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.62.7.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.62.7.2 compute

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.62.7.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.62.7.2.2 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

7.2.62.7.2.3 system

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.62.7.2.4 series

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.62.7.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [response_levels](#)

- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.62.8 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not

very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means = 248.89, 593.33
  std_deviations = 12.4, 29.7
  descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds = 199.3, 474.63
  upper_bounds = 298.5, 712.
  descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas = 12., 30.
  betas = 250., 590.
  descriptors = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs = 2
  abscissas = 3 4
  counts = 1 1
  descriptors = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

```
PDF for response_fn_1:
  Bin Lower          Bin Upper          Density Value
  -----
  2.7604749078e+11  3.4221494996e+11  5.1384774972e-12
  3.4221494996e+11  4.0634975300e+11  5.1454122311e-12
  4.0634975300e+11  5.4196114379e+11  2.4334239039e-12
PDF for response_fn_2:
  Bin Lower          Bin Upper          Density Value
  -----
  4.6431154744e+04  5.6511827775e+04  1.9839945149e-05
  5.6511827775e+04  6.1603813790e+04  5.8916108390e-05
  6.1603813790e+04  7.8702465755e+04  2.9242071306e-05
PDF for response_fn_3:
```

Bin Lower -----	Bin Upper -----	Density Value -----
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level -----	Probability Level -----	Reliability Index -----	General Rel Index -----
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level -----	Probability Level -----	Reliability Index -----	General Rel Index -----
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level -----	Probability Level -----	Reliability Index -----	General Rel Index -----
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.62.8.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.62.9 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.62.9.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [gpais](#)

- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.62.10 distribution

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword	Dakota Keyword Description
			cumulative	Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.62.10.1 cumulative

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.62.10.2 complementary

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.62.11 rng

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.62.11.1 mt19937

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.62.11.2 rnum2

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.62.12 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [gpais](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```
    surrogate global,  
    dace_method_pointer = 'DACE'  
    polynomial quadratic  
  
method  
  id_method = 'DACE'  
  model_pointer = 'DACE_M'  
  sampling sample_type lhs  
  samples = 121 seed = 5034 rng rnum2  
  
model  
  id_model = 'DACE_M'  
  single  
  interface_pointer = 'I1'  
  
variables  
  uniform_uncertain = 2  
  lower_bounds = 0.  0.  
  upper_bounds = 1.  1.  
  descriptors = 'x1' 'x2'  
  
interface  
  id_interface = 'I1'  
  system async evaluation_concurrency = 5  
  analysis_driver = 'text_book'  
  
responses  
  response_functions = 3  
  no_gradients  
  no_hessians
```

7.2.63 adaptive_sampling

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)

(Experimental) Adaptively refine a Gaussian process surrogate

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)

Specification

Alias: nond_adaptive_sampling

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_samples	Initial number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional		samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional		fitness_metric	(Experimental) Specify the <code>fitness_metric</code> used to select the next point
	Optional		batch_selection	(Experimental) How to select new points
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		misc_options	(Experimental) Hook for algorithm-specific adaptive sampling options

	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions
	Optional	rng	Selection of a random number generator
	Optional	model_pointer	Identifier for model block to be used by a method

Description

This is an experimental capability that is not ready for production use at this point. It was part of an investigation into computational topology-based approaches to feature identification and surrogate refinement.

The goal in performing adaptive sampling is to construct a surrogate model that can be used as an accurate predictor to some expensive simulation, thus it is to one's advantage to build a surrogate that minimizes the error over the entire domain of interest using as little data as possible from the expensive simulation. The adaptive part alludes to the fact that the surrogate will be refined by focusing samples of the expensive simulation on particular areas of interest rather than rely on random selection or standard space-filling techniques.

At a high-level, the adaptive sampling pipeline is a four-step process:

- Evaluate the expensive simulation (referred to as the true model) at initial sample point
 1. Fit a surrogate model
 2. Create a candidate set and score based on information from surrogate
 3. Select a candidate point to evaluate the true model
 4. Loop until done

In terms of the Dakota implementation, the adaptive sampling method currently uses Latin Hypercube sampling (LHS) to generate the initial points in Step 1 above. For Step 2, we use a Gaussian process model.

The default behavior is to add one point at a time. At each iteration (e.g. each loop of Steps 2-4 above), a Latin Hypercube sample is generated (a new one, different from the initial sample) and the surrogate model is evaluated at this points. These are the candidate points that are then evaluated according to the fitness metric. The number of candidates used in practice should be high enough to fill most of the input domain: we recommend at least hundreds of points for a low-dimensional problem. All of the candidates (samples on the emulator) are given a score and then the highest-scoring candidate is selected to be evaluated on the true model.

The adaptive sampling method also can generate batches of points to add at a time using the `batch_selection` and `batch_size` keywords.

See Also

These keywords may also be of interest:

- [gpais](#)
- [local_reliability](#)
- [global_reliability](#)
- [sampling](#)
- [importance_sampling](#)
- [polynomial_chaos](#)
- [stoch_collocation](#)

7.2.63.1 initial_samples

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [initial_samples](#)

Initial number of samples for sampling-based methods

Specification

Alias: samples

Argument(s): INTEGER

Default: 0

Description

The `initial_samples` keyword is used to define the number of initial samples (i.e., randomly chosen sets of variable values) at which to execute a model. The initial samples may later be augmented in an iterative process.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least

$(dim+1)(dim+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10 \cdot dim$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N \cdot (dim+2)$.

Examples

```
method
  sampling
    sample_type random
    initial_samples = 20
    refinement_samples = 5
```

7.2.63.2 seed

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```


7.2.63.3 `samples_on_emulator`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: none

Argument(s): INTEGER

Default: 400

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.63.4 `fitness_metric`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [fitness_metric](#)

(Experimental) Specify the `fitness_metric` used to select the next point

Specification

Alias: none

Argument(s): none

Default: `predicted_variance`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Fitness Metric (Group 1)	predicted_variance	Pick points with highest variance
			distance	Space filling metric
			gradient	Fill the range space of the surrogate

Description

`adaptive` sampling is an experimental capability that is not ready for production use at this time.

The user can specify the `fitness_metric` used to select the next point (or points) to evaluate and add to the set. The fitness metrics used for scoring candidate points include:

- `predicted_variance`
- `distance`
- `gradient`

7.2.63.4.1 `predicted_variance`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [fitness_metric](#)
- [predicted_variance](#)

Pick points with highest variance

Specification

Alias: none

Argument(s): none

Description

The predicted variance metric uses the predicted variance of the Gaussian process surrogate as the score of a candidate point. Thus, the adaptively chosen points will be in areas of highest uncertainty according to the Gaussian process model.

7.2.63.4.2 `distance`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [fitness_metric](#)
- [distance](#)

Space filling metric

Specification

Alias: none

Argument(s): none

Description

The distance metric calculates the Euclidean distance in domain space between the candidate and its nearest neighbor in the set of points already evaluated on the true model. Therefore, the most undersampled area of the domain will always be selected. Note that this is a space-filling metric.

7.2.63.4.3 gradient

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [fitness_metric](#)
- [gradient](#)

Fill the range space of the surrogate

Specification

Alias: none

Argument(s): none

Description

The gradient metric calculates the score as the absolute value of the difference in range space (the outputs) of the two points. The output values used are predicted from the surrogate model. This method attempts to evenly fill the range space of the surrogate.

7.2.63.5 batch_selection

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [batch_selection](#)

(Experimental) How to select new points

Specification

Alias: none

Argument(s): none

Default: naive

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (Choose One)	Batch Selection Criterion (Group 1)	naive	Take the highest scoring candidates
			distance_penalty	Add a penalty to spread out the points in the batch
			topology	In this selection strategy, we use information about the topology of the space from the Morse-Smale complex to identify next points to select.
			constant_liar	Use information from the existing surrogate model to predict what the surrogate upgrade will be with new points.

Description

`adaptive_sampling` is an experimental capability that is not ready for production use at this time.

With batch or multi-point selection, the true model can be evaluated in parallel and thus increase throughput before refitting our surrogate model. This proposes a new challenge as the problem of choosing a single point and choosing multiple points off a surrogate are fundamentally different. Selecting the n best scoring candidates is more than likely to generate a set of points clustered in one area which will not be conducive to adapting the surrogate.

We have implemented several strategies for batch selection of points. These are described in the User's manual and are the subject of active research.

The `batch_selection` strategies include:

1. `naive`:
2. `distance_penalty`
3. `constant_liar`
4. `topology`

7.2.63.5.1 naive

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [batch_selection](#)
- [naive](#)

Take the highest scoring candidates

Specification

Alias: none

Argument(s): none

Description

This strategy will select the n highest scoring candidates regardless of their position. This tends to group an entire round of points in the same area.

7.2.63.5.2 distance_penalty

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [batch_selection](#)
- [distance_penalty](#)

Add a penalty to spread out the points in the batch

Specification

Alias: none

Argument(s): none

Description

In this strategy, the highest scoring candidate is selected and then all remaining candidates are re-scored with a distance penalization factor added in to the score.

7.2.63.5.3 topology

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [batch_selection](#)
- [topology](#)

In this selection strategy, we use information about the topology of the space from the Morse-Smale complex to identify next points to select.

Specification

Alias: none

Argument(s): none

Description

In this strategy we look at the topology of the scoring function and select the n highest maxima in the topology. To determine local maxima, we construct the approximate Morse-Smale complex. This strategy does require the user to have the Morse-Smale package.

7.2.63.5.4 constant_liar

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [batch_selection](#)
- [constant_liar](#)

Use information from the existing surrogate model to predict what the surrogate upgrade will be with new points.

Specification

Alias: none

Argument(s): none

Description

The strategy first selects the highest scoring candidate, and then refits the surrogate using a "lie" value at the point selected. The 'lie' value is based on the surrogate predictions and not the simulation. This process repeats until n points have been selected whereupon the lie values are removed from the surrogate and the selected points are evaluated on the true model and the surrogate is refit with these values.

7.2.63.6 refinement_samples

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the initial_samples in a sampling design.

7.2.63.7 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.63.7.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.63.7.1.1 header

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.63.7.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.63.7.1.3 interface_id

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.63.7.2 annotated

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.63.7.3 freeform

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.63.7.4 active_only

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.63.8 export_approx_points_file

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: export_points_file

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.63.8.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002      0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.63.8.1.1 header

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.63.8.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.63.8.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.63.8.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002      0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.63.8.3 freeform

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
    0.90009        1.1 0.0001996404857  0.2601620081  0.759955
    0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.63.9 misc_options

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [misc_options](#)

(Experimental) Hook for algorithm-specific adaptive sampling options

Specification

Alias: none

Argument(s): STRINGLIST

Default: no misc options

Description

The adaptive sampling algorithm is an experimental capability and not ready for production use at this time.

7.2.63.10 max_iterations

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.63.11 response_levels

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12

```

3.6000000000e+11 4.0000000000e+11 4.2500000000e-12
4.0000000000e+11 4.4000000000e+11 3.7500000000e-12
4.4000000000e+11 5.4196114379e+11 2.2557612778e-12
PDF for response_fn_2:
  Bin Lower          Bin Upper          Density Value
  -----
4.6431154744e+04 6.0000000000e+04 2.8742313192e-05
6.0000000000e+04 6.5000000000e+04 6.4000000000e-05
6.5000000000e+04 7.0000000000e+04 4.0000000000e-05
7.0000000000e+04 7.8702465755e+04 1.0341896485e-05
PDF for response_fn_3:
  Bin Lower          Bin Upper          Density Value
  -----
2.3796737090e+05 3.5000000000e+05 4.2844660868e-06
3.5000000000e+05 4.0000000000e+05 8.6000000000e-06
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06

```

Level mappings for each response function:

```

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
3.6000000000e+11 5.5000000000e-01
4.0000000000e+11 3.8000000000e-01
4.4000000000e+11 2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
6.0000000000e+04 6.1000000000e-01
6.5000000000e+04 2.9000000000e-01
7.0000000000e+04 9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
3.5000000000e+05 5.2000000000e-01
4.0000000000e+05 9.0000000000e-02
4.5000000000e+05 0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.63.11.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.63.11.2 compute

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels

			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.63.11.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.63.11.2.2 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

7.2.63.11.2.3 system

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.63.11.2.4 series

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.63.11.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.63.12 probability_levels

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	Specify which probability_levels correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
```

```

uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors        = 'TF1u' 'TF2u'
weibull_uncertain = 2
  alphas            = 12., 30.
  betas              = 250., 590.
  descriptors        = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
  num_pairs         = 3      4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors        = 'TF1h' 'TF2h'
histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors        = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		

```
3.6997214153e+05 3.0000000000e-01
3.8100966235e+05 2.0000000000e-01
4.4111498127e+05 0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.63.12.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page

7.2.63.13 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.63.13.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.63.14 distribution

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
```

```
sample_type lhs
samples = 10
distribution cumulative
```

7.2.63.14.1 cumulative

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.63.14.2 complementary

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.63.15 rng

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister

			<code>rnum2</code>	Generates pseudo-random numbers using the Pecos package
--	--	--	--------------------	---

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.63.15.1 mt19937

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.63.15.2 rnum2

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.63.16 model_pointer

- [Keywords Area](#)
- [method](#)
- [adaptive_sampling](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.64 pof_darts

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)

Probability-of-Failure (POF) darts is a novel method for estimating the probability of failure based on random sphere-packing.

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)

Specification

Alias: nond_pof_darts

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		build_samples	Number of initial model evaluations used in build phase
	Optional		seed	Seed of the random number generator
	Optional		lipschitz	Select the type of Lipschitz estimation (global or local)
	Optional		samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)

	Optional	response_levels	Values at which to estimate desired statistics for each response
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions
	Optional	rng	Selection of a random number generator
	Optional	model_pointer	Identifier for model block to be used by a method

Description

`pof_darts` is a novel method for estimating the probability of failure based on random sphere-packing. Random spheres are sampled from the domain with the constraint that each new sphere center has to be outside prior disks. The radius of each sphere is chosen such that the entire sphere lie either in the failure or the non-failure region. This radius depends of the function evaluation at the disk center, the failure threshold and an estimate of the function gradient at the disk center.

We utilize a global surrogate for evaluating the gradient and hence only one function evaluation is required for each sphere.

After exhausting the sampling budget specified by `samples`, which is the number of spheres per failure threshold, the domain is decomposed into two regions. These regions correspond to failure and non-failure, each represented by the union of the spheres of each type. The volume of the union of failure spheres gives a lower bound on the required estimate of the probability of failure, while the volume of the union of the non-failure spheres subtracted from the volume of the domain gives an upper estimate. We currently report the average of both estimates.

`pof_darts` handles multiple response functions and allows each to have multiple failure thresholds. For each failure threshold, `pof_darts` will insert a number of spheres specified by the user-input parameter `samples`.

However, estimating the probability of failure for each failure threshold would utilize the total number of disks sampled for all failure thresholds. For each failure threshold, the sphere radii changes to generate the right spatial decomposition.

7.2.64.1 `build_samples`

- [Keywords Area](#)
- [method](#)

- [pof_darts](#)
- [build_samples](#)

Number of initial model evaluations used in build phase

Specification

Alias: samples

Argument(s): INTEGER

Description

The number of build points or training points used in the initial phase of an algorithm or model construction. Typically these are the initial set of data points used to construct (train) a surrogate model (emulator). If the number of `build_samples` is less than the minimum number of points required to build the surrogate, Dakota will augment the samples to obtain the minimum required.

Examples

Perform GP-based adaptive importance sampling, building the GP with 100 points and then performing 100 approximate evaluations to evaluate the probability.

```
method
  gpais
    build_samples = 100
    samples_on_emulator = 100
    max_iterations = 5
    response_levels = -1.065
```

7.2.64.2 seed

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.64.3 lipschitz

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [lipschitz](#)

Select the type of Lipschitz estimation (global or local)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Local/Global Estimate (Group 1)	Dakota Keyword local	Dakota Keyword Description Specify local estimation of the Lipschitz constant
			global	Specify global estimation of the Lipschitz estimate

Description

There are two types of Lipschitz estimation used in sizing the disks used in POF Darts: `global` and `local`. The global approach uses one Lipschitz estimate for the entire domain. The local approach calculates the Lipschitz estimate separately for each Voronoi region based on nearby points. The local method is more expensive but more accurate.

7.2.64.3.1 local

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [lipschitz](#)
- [local](#)

Specify local estimation of the Lipschitz constant

Specification

Alias: none

Argument(s): none

Description

The local approach to estimate the Lipschitz constant calculates the Lipschitz estimate separately for each Voronoi region based on nearby points. The local method is more expensive but results in higher accuracy compared to the global method.

7.2.64.3.2 global

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [lipschitz](#)
- [global](#)

Specify global estimation of the Lipschitz estimate

Specification

Alias: none

Argument(s): none

Description

The global approach uses one Lipschitz estimate for the entire domain. This option is currently deactivated. Please refer to the local alternative.

7.2.64.4 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)

- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: none

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.64.5 response_levels

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response

	Optional	<code>compute</code>	Selection of statistics to compute at each response level
--	-----------------	----------------------	---

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
```

```

    upper_bounds      = 298.5, 712.
    descriptors       = 'TF1u' 'TF2u'
weibull_uncertain = 2
    alphas           = 12., 30.
    betas            = 250., 590.
    descriptors       = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
    num_pairs        = 3 4
    abscissas        = 5 8 10 .1 .2 .3 .4
    counts           = 17 21 0 12 24 12 0
    descriptors       = 'TF1h' 'TF2h'
histogram_point_uncertain
    real = 1
    num_pairs        = 2
    abscissas        = 3 4
    counts           = 1 1
    descriptors       = 'TF3h'

interface,
    system asynch evaluation_concurrency = 5
    analysis_driver = 'text_book'

responses,
    response_functions = 3
    no_gradients
    no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06
4.0000000000e+05	4.5000000000e+05	1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
3.6000000000e+11	5.5000000000e-01		
4.0000000000e+11	3.8000000000e-01		
4.4000000000e+11	2.3000000000e-01		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
6.0000000000e+04	6.1000000000e-01		
6.5000000000e+04	2.9000000000e-01		
7.0000000000e+04	9.0000000000e-02		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
3.5000000000e+05	5.2000000000e-01		
4.0000000000e+05	9.0000000000e-02		
4.5000000000e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.64.5.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.64.5.2 compute

- [Keywords Area](#)
- [method](#)

- [pof_darts](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, `probabilities` will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
```



```

sample_type random
samples = 100 seed = 1
complementary distribution
response_levels = 3.6e+11 4.0e+11 4.4e+11
                  6.0e+04 6.5e+04 7.0e+04
                  3.5e+05 4.0e+05 4.5e+05
compute reliabilities

```

7.2.64.5.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```

method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities

```

7.2.64.5.2.2 gen_reliabilities

- [Keywords Area](#)
- [method](#)

- [pof_darts](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

7.2.64.5.2.3 system

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.64.5.2.4 series

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.64.5.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [response_levels](#)

- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.64.6 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not

very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
    1. .8 .5 0.
    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means = 248.89, 593.33
  std_deviations = 12.4, 29.7
  descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds = 199.3, 474.63
  upper_bounds = 298.5, 712.
  descriptors = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas = 12., 30.
  betas = 250., 590.
  descriptors = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs = 3 4
  abscissas = 5 8 10 .1 .2 .3 .4
  counts = 17 21 0 12 24 12 0
  descriptors = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs = 2
  abscissas = 3 4
  counts = 1 1
  descriptors = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower -----	Bin Upper -----	Density Value -----
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level -----	Probability Level -----	Reliability Index -----	General Rel Index -----
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level -----	Probability Level -----	Reliability Index -----	General Rel Index -----
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level -----	Probability Level -----	Reliability Index -----	General Rel Index -----
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.64.6.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.64.7 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.64.7.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)

- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.64.8 distribution

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword	Dakota Keyword Description
			cumulative	Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.64.8.1 cumulative

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.64.8.2 complementary

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.64.9 rng

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.64.9.1 mt19937

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.64.9.2 rnum2

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.64.10 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [pof_darts](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```

    surrogate global,
    dace_method_pointer = 'DACE'
    polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.65 rkd_darts

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)

Recursive k-d (RKD) Darts: Recursive Hyperplane Sampling for Numerical Integration of High-Dimensional Functions.

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)

Specification

Alias: nond_rkd_darts

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		build_samples	Number of initial model evaluations used in build phase

	Optional	seed	Seed of the random number generator
	Optional	lipschitz	Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
	Optional	samples_on_emulator	Number of samples at which to evaluate an emulator (surrogate)
	Optional	response_levels	Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
	Optional	probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional	gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional	distribution	Selection of cumulative or complementary cumulative functions
	Optional	rng	Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
	Optional	model_pointer	Identifier for model block to be used by a method

Description

Disclaimer: The RKD method is currently in development mode, for further experimental verification. Please contact Dakota team if you have further questions about using this method.

Recursive k-d (RKD) darts is an algorithm to evaluate the integration of a d-dimensional black box function $f(x)$ via recursive sampling over d , using a series of hyperplanes of variable dimensionality $k = \{d, d-1, \dots, 0\}$. Fundamentally, we decompose the d-dimensional integration problem into a series of nested one-dimensional problems. That is, we start at the root level (the whole domain) and start sampling down using hyperplanes of one lower dimension, all the way down to zero (points). A d-dimensional domain is subsampled using (d-1) hyperplanes, a (d-1)-dimensional sub-domain is subsampled using (d-2) hyperplanes, and so on. Every hyperplane, regardless of its dimension, is evaluated using sampled hyperplanes of one lower dimension, as shown in the set of figures above. Each hyperplane has direct information exchange with its parent hyperplane of one higher dimension, and its children of one lower dimension.

In each one-dimensional problem, we construct a piecewise approximation surrogate model, using 1-dimensional Lagrange interpolation. Information is exchanged between different levels, including integration values, as well as interpolation and evaluation errors, in order to a) find the integration value up to that level, b) estimate the associated integration error, and c) guide the placement of future samples.

7.2.65.1 `build_samples`

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [build_samples](#)

Number of initial model evaluations used in build phase

Specification

Alias: `samples`

Argument(s): INTEGER

Description

The number of build points or training points used in the initial phase of an algorithm or model construction. Typically these are the initial set of data points used to construct (train) a surrogate model (emulator). If the number of `build_samples` is less than the minimum number of points required to build the surrogate, Dakota will augment the samples to obtain the minimum required.

Examples

Perform GP-based adaptive importance sampling, building the GP with 100 points and then performing 100 approximate evaluations to evaluate the probability.

```
method
  gpais
    build_samples = 100
    samples_on_emulator = 100
    max_iterations = 5
    response_levels = -1.065
```

7.2.65.2 `seed`

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.65.3 lipschitz

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [lipschitz](#)

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Local/Global Estimate (Group 1)	local	Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
			global	Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

7.2.65.3.1 local

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [lipschitz](#)
- [local](#)

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none

Argument(s): none

7.2.65.3.2 global

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [lipschitz](#)
- [global](#)

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none

Argument(s): none

7.2.65.4 samples_on_emulator

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)

- [samples_on_emulator](#)

Number of samples at which to evaluate an emulator (surrogate)

Specification

Alias: none

Argument(s): INTEGER

Description

How many approximate function evaluations to perform on the emulator model, e.g., to compute statistics

Default Behavior

The default number of samples is method-dependent.

Examples

Perform 10000 samples on the PCE approximation of the true model:

```
method
  polynomial_chaos
    quadrature_order = 2
    samples_on_emulator = 10000
```

7.2.65.5 response_levels

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

7.2.65.5.1 [num_response_levels](#)

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: [response_levels](#) evenly distributed among response functions

Description

The [num_response_levels](#) keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If [num_response_levels](#) is not specified, the [response_levels](#) will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.65.5.2 compute

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, `probabilities` will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.65.5.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                     6.0e+04 6.5e+04 7.0e+04
                     3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.65.5.2.2 gen_reliabilities

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute gen_reliabilities
```

7.2.65.5.2.3 system

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group System Reliability Mode (Group 1)	Dakota Keyword	Dakota Keyword Description
			series	Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.65.5.2.4 [series](#)

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.65.5.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.65.6 probability_levels

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	Specify which probability_levels correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds    = 199.3, 474.63
  upper_bounds    = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs       = 3 4
  abscissas       = 5 8 10 .1 .2 .3 .4
  counts          = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs       = 2
  abscissas       = 3 4
  counts          = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:
PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		
3.6997214153e+05	3.0000000000e-01		
3.8100966235e+05	2.0000000000e-01		
4.4111498127e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.65.6.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page

7.2.65.7 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.65.7.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.65.8 distribution

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.65.8.1 cumulative

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.65.8.2 complementary

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.65.9 rng

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [rng](#)

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification**Alias:** none**Argument(s):** none**Default:** Mersenne twister (mt19937)**Child Keywords:**

	Required/Optional <i>Required(Choose One)</i>	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Undocumented: Recursive k-d (RKD) Darts is an experimental capability.
			rnum2	Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

7.2.65.9.1 mt19937

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [rng](#)
- [mt19937](#)

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification**Alias:** none**Argument(s):** none**7.2.65.9.2 rnum2**

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [rng](#)
- [rnum2](#)

Undocumented: Recursive k-d (RKD) Darts is an experimental capability.

Specification**Alias:** none**Argument(s):** none

7.2.65.10 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [rkd_darts](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```
    surrogate global,  
    dace_method_pointer = 'DACE'  
    polynomial quadratic  
  
method  
  id_method = 'DACE'  
  model_pointer = 'DACE_M'  
  sampling sample_type lhs  
  samples = 121 seed = 5034 rng rnum2  
  
model  
  id_model = 'DACE_M'  
  single  
  interface_pointer = 'I1'  
  
variables  
  uniform_uncertain = 2  
  lower_bounds = 0.  0.  
  upper_bounds = 1.  1.  
  descriptors = 'x1' 'x2'  
  
interface  
  id_interface = 'I1'  
  system async evaluation_concurrency = 5  
  analysis_driver = 'text_book'  
  
responses  
  response_functions = 3  
  no_gradients  
  no_hessians
```

7.2.66 global_evidence

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)

Evidence theory with evidence measures computed with global optimization methods

Topics

This keyword is related to the topics:

- [epistemic_uncertainty_quantification_methods](#)
- [evidence_theory](#)

Specification

Alias: nond_global_evidence

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		samples	Number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional (<i>Choose One</i>)	Solution Approach (Group 1)	sbgo	Use the surrogate based optimization method
			ego	Use the Efficient Global Optimization method
			ea	Use an evolutionary algorithm
			lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		rng	Selection of a random number generator
	Optional		model_pointer	Identifier for model block to be used by a method

Description

`global_evidence` allows the user to specify several global approaches for calculating the belief and plausibility functions:

- `lhs` - note: this takes the minimum and maximum of the samples as the bounds per "interval cell combination."
- `ego` - uses Efficient Global Optimization which is based on an adaptive Gaussian process surrogate.
- `sbo` - uses a Gaussian process surrogate (non-adaptive) within an optimization process.
- `ea` - uses an evolutionary algorithm. This can be expensive as the `ea` will be run for each interval cell combination.

Note that to calculate the plausibility and belief cumulative distribution functions, one has to look at all combinations of intervals for the uncertain variables. In terms of implementation, if one is using LHS sampling as outlined above, this method creates a large sample over the response surface, then examines each cell to determine the minimum and maximum sample values within each cell. To do this, one needs to set the number of samples relatively high: the default is 10,000 and we recommend at least that number. If the model you are running is a simulation that is computationally quite expensive, we recommend that you set up a surrogate model within the Dakota input file so that `global_evidence` performs its sampling and calculations on the surrogate and not on the original model. If one uses optimization methods instead to find the minimum and maximum sample values within each cell, this can also be computationally expensive.

Additional Resources

See the topic page [evidence_theory](#) for important background information and usage notes.

Refer to [variable_support](#) for information on supported variable types.

Theory

The basic idea is that one specifies an "evidence structure" on uncertain inputs and propagates that to obtain belief and plausibility functions on the response functions. The inputs are defined by sets of intervals and Basic Probability Assignments (BPAs). Evidence propagation is computationally expensive, since the minimum and maximum function value must be calculated for each "interval cell combination." These bounds are aggregated into belief and plausibility.

See Also

These keywords may also be of interest:

- [global_interval_est](#)
- [local_evidence](#)
- [local_interval_est](#)

7.2.66.1 samples

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)

- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10*\text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N*(\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.66.2 seed

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.66.3 sbgo

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)

Use the surrogate based optimization method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		gaussian_process	Gaussian Process surrogate model
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

	Optional	export_approx_points_file	Output file for surrogate model value evaluations
--	-----------------	---	---

Description

A surrogate-based optimization method will be used. The surrogate employed in `sbo` is a Gaussian process surrogate.

The main difference between `ego` and the `sbo` approach is the objective function being optimized. `ego` relies on an expected improvement function, while in `sbo`, the optimization proceeds using an evolutionary algorithm ([coliny_ea](#)) on the Gaussian process surrogate: it is a standard surrogate-based optimization. Also note that the `sbo` option can support optimization over discrete variables (the discrete variables are relaxed) while `ego` cannot.

This is not the same as [surrogate_based_global](#).

7.2.66.3.1 gaussian_process

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group GP Implementation (Group 1)	Dakota Keyword	Dakota Keyword Description
			surfpack	Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate
			experimental	Use the experimental Gaussian Process surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.66.3.1.1 surfpack

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.66.3.1.2 dakota

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.66.3.1.3 experimental

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [gaussian_process](#)
- [experimental](#)

Use the experimental Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		options_file	Filename for a YAML file that specifies Gaussian process options

Description

The experimental GP surrogate is an emerging capability intended to supersede the `dakota` and `surfpack` GP implementations by unifying the best features of both. It also includes additional features such as the ability to export the model and then use it from Python.

See Also

These keywords may also be of interest:

- [experimental_gaussian_process](#)

7.2.66.3.1.4 options_file

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [gaussian_process](#)
- [experimental](#)
- [options_file](#)

Filename for a YAML file that specifies Gaussian process options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the Gaussian process that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: gp_options.yaml

GP Parameters:
  Sigma Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  scaler name: standardization
  num restarts: 20
  Nugget:
    fixed nugget: 0.0
    estimate nugget: true
  Bounds:
    lower bound: 3.17e-8
    upper bound: 1.0e-2
  gp seed: 42
  standardize response: false
  Trend:
    estimate trend: true
  Options:
    max degree: 2
    reduced basis: false
    p-norm: 1.0
    scaler type: none
```

```

    regression solver type: SVD
    verbosity: 1
kernel type: squared exponential
Length-scale Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
verbosity: 1

```

7.2.66.3.2 use_derivatives

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.66.3.3 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.66.3.3.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.3.3.2 header

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.66.3.3.3 eval_id

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.66.3.3.4 interface_id

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.66.3.3.5 annotated

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009       1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991       1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.3.3.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.3.3.7 active_only

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.66.3.4 export_approx_points_file

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)

- [sbgo](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: export_points_file

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.66.3.4.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.3.4.2 header

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.66.3.4.3 eval_id

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.66.3.4.4 interface_id

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.66.3.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009       1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991       1.1 0.0002003604863 0.2598380081 0.760045
...
```


7.2.66.3.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.4 ego

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)

Use the Efficient Global Optimization method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		gaussian_process	Gaussian Process surrogate model
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations

Description

In the case of `ego`, the efficient global optimization (EGO) method is used to calculate bounds. By default, the Surfpack GP (Kriging) model is used, but the Dakota implementation may be selected instead. If `use_derivatives` is specified the GP model will be built using available derivative data (Surfpack GP only).

See [efficient_global](#) for more information.

7.2.66.4.1 gaussian_process

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional	Description of Group GP Implementation (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		surfpack	Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate
			experimental	Use the experimental Gaussian Process surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.66.4.1.1 surfpack

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	export_model not active

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.66.4.1.2 export_model

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)

export_model not active

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	export_model not active
	Required		formats	export_model not active

Description

Keyword reserved for future use

7.2.66.4.1.3 filename_prefix

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [filename_prefix](#)

export_model not active

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Keyword reserved for future use

7.2.66.4.1.4 formats

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)

export_model not active

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	export_model not active
	Optional		binary_archive	export_model not active

Description

Keyword reserved for future use

7.2.66.4.1.5 [text_archive](#)

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

export_model not active

Specification

Alias: none

Argument(s): none

Description

Keyword reserved for future use

7.2.66.4.1.6 [binary_archive](#)

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)

- [binary_archive](#)

export_model not active

Specification

Alias: none

Argument(s): none

Description

Keyword reserved for future use

7.2.66.4.1.7 dakota

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.66.4.1.8 experimental

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)

Use the experimental Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	export_model not active
	Optional		options_file	options_file not active

Description

The experimental GP surrogate is an emerging capability intended to supersede the `dakota` and `surfpack` GP implementations by unifying the best features of both. It also includes additional features such as the ability to export the model and then use it from Python.

See Also

These keywords may also be of interest:

- [experimental_gaussian_process](#)

7.2.66.4.1.9 export_model

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)

export_model not active

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	export_model not active
	Required		formats	export_model not active

Description

Keyword reserved for future use

7.2.66.4.1.10 filename_prefix

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [filename_prefix](#)

export_model not active

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Keyword reserved for future use

7.2.66.4.1.11 formats

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)

- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [formats](#)

export_model not active

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	export_model not active
	Optional		binary_archive	export_model not active

Description

Keyword reserved for future use

7.2.66.4.1.12 [text_archive](#)

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

export_model not active

Specification

Alias: none

Argument(s): none

Description

Keyword reserved for future use

7.2.66.4.1.13 `binary_archive`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

`export_model` not active

Specification

Alias: none

Argument(s): none

Description

Keyword reserved for future use

7.2.66.4.1.14 `options_file`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [options_file](#)

`options_file` not active

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

Keyword reserved for future use

7.2.66.4.2 use_derivatives

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.66.4.3 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format

		annotated	Selects annotated tabular file format
		freeform	Selects freeform file format
	Optional	active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.66.4.3.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

%eval_id	x1	x2	obj_fn	nln_ineq_con_1	nln_ineq_con_2
1	0.9	1.1	0.0002	0.26	0.76
2	0.90009	1.1	0.0001996404857	0.2601620081	0.759955
3	0.89991	1.1	0.0002003604863	0.2598380081	0.760045
...					

7.2.66.4.3.2 header

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.66.4.3.3 eval_id

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.66.4.3.4 `interface_id`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.66.4.3.5 `annotated`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002      0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.4.3.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009      1.1 0.0001996404857  0.2601620081  0.759955
0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.66.4.3.7 active_only

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.66.4.4 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
--	--	--	--------------------------	------------------------------

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.66.4.4.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.66.4.4.2 header

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)

- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.66.4.4.3 `eval_id`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.66.4.4.4 `interface_id`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [export_approx_points_file](#)

- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.66.4.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.4.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ego](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.66.5 ea

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [ea](#)

Use an evolutionary algorithm

Specification

Alias: none

Argument(s): none

Description

In this approach, the evolutionary algorithm from Coliny, [coliny_ea](#), is used to perform the interval optimization with no surrogate model involved. Again, this option of `ea` can support interval optimization over discrete variables.

7.2.66.6 lhs

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.66.7 response_levels

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_response_levels</code>	Number of values at which to estimate desired statistics for each response
	Optional		<code>compute</code>	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num-`

`_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors     = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds    = 199.3, 474.63
  upper_bounds    = 298.5, 712.
  descriptors     = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors     = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs      = 3 4
  abscissas     = 5 8 10 .1 .2 .3 .4
  counts        = 17 21 0 12 24 12 0
  descriptors   = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs    = 2
  abscissas   = 3 4
  counts      = 1 1
  descriptors = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06

```
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06
```

Level mappings for each response function:

```
Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.6000000000e+11  5.5000000000e-01
  4.0000000000e+11  3.8000000000e-01
  4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  6.0000000000e+04  6.1000000000e-01
  6.5000000000e+04  2.9000000000e-01
  7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
  3.5000000000e+05  5.2000000000e-01
  4.0000000000e+05  9.0000000000e-02
  4.5000000000e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.66.7.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.66.7.2 compute

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Statistics to Compute (Group 1)	Dakota Keyword probabilities	Dakota Keyword Description Computes probabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels

	Optional	system	Compute system reliability (series or parallel)
--	-----------------	------------------------	---

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.66.7.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.66.7.2.2 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

7.2.66.7.2.3 system

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.66.7.2.4 series

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.66.7.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.66.8 probability_levels

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	Specify which probability_levels correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
```

```

uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
  num_pairs        = 3      4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors       = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		

3.6997214153e+05	3.0000000000e-01
3.8100966235e+05	2.0000000000e-01
4.4111498127e+05	0.0000000000e+00

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.66.8.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.66.9 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.66.9.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.66.10 distribution

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
```



```
sample_type lhs
samples = 10
distribution cumulative
```

7.2.66.10.1 cumulative

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.66.10.2 complementary

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.66.11 rng

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister

			<code>rnum2</code>	Generates pseudo-random numbers using the Pecos package
--	--	--	--------------------	---

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.66.11.1 mt19937

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.66.11.2 rnum2

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.66.12 model_pointer

- [Keywords Area](#)
- [method](#)
- [global_evidence](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```
variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.67 `global_interval_est`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)

Interval analysis using global optimization methods

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)
- [epistemic_uncertainty_quantification_methods](#)
- [interval_estimation](#)

Specification

Alias: `nond_global_interval_est`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		samples	Number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional (<i>Choose One</i>)	Solution Approach (Group 1)	sbgo	Use the surrogate based optimization method
			ego	Use the Efficient Global Optimization method
			ea	Use an evolutionary algorithm
			lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
	Optional		rng	Selection of a random number generator
	Optional		model_pointer	Identifier for model block to be used by a method

Description

In the global approach to interval estimation, one uses either a global optimization method or a sampling method to assess the bounds of the responses.

`global_interval_est` allows the user to specify several approaches to calculate interval bounds on the output responses.

- `lhs` - note: this takes the minimum and maximum of the samples as the bounds
- `ego`

- [sbo](#)
- [ea](#)

Additional Resources

Refer to [variable_support](#) for information on supported variable types.

See Also

These keywords may also be of interest:

- [global_evidence](#)
- [local_evidence](#)
- [local_interval_est](#)

7.2.67.1 `samples`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10*\text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N*(\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```


7.2.67.2 seed

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.67.3 max_iterations

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.67.4 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.67.5 `max_function_evaluations`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.2.67.6 sbgo

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)

Use the surrogate based optimization method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		gaussian_process	Gaussian Process surrogate model
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations

Description

A surrogate-based optimization method will be used. The surrogate employed in `sbo` is a Gaussian process surrogate.

The main difference between `ego` and the `sbo` approach is the objective function being optimized. `ego` relies on an expected improvement function, while in `sbo`, the optimization proceeds using an evolutionary algorithm ([coliny_ea](#)) on the Gaussian process surrogate: it is a standard surrogate-based optimization. Also note that the `sbo` option can support optimization over discrete variables (the discrete variables are relaxed) while `ego` cannot.

This is not the same as [surrogate_based_global](#).

7.2.67.6.1 gaussian_process

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	GP Implementation (Group 1)	surfpack	Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate
			experimental	Use the experimental Gaussian Process surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.67.6.1.1 surfpack

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [gaussian_process](#)

- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use

derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.67.6.1.2 dakota

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the `dakota` version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.67.6.1.3 experimental

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [gaussian_process](#)
- [experimental](#)

Use the experimental Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		options_file	Filename for a YAML file that specifies Gaussian process options

Description

The experimental GP surrogate is an emerging capability intended to supersede the `dakota` and `surfpack` GP implementations by unifying the best features of both. It also includes additional features such as the ability to export the model and then use it from Python.

See Also

These keywords may also be of interest:

- [experimental_gaussian_process](#)

7.2.67.6.1.4 options_file

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [gaussian_process](#)
- [experimental](#)
- [options_file](#)

Filename for a YAML file that specifies Gaussian process options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the Gaussian process that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: gp_options.yaml

GP Parameters:
  Sigma Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  scaler name: standardization
  num restarts: 20
  Nugget:
    fixed nugget: 0.0
    estimate nugget: true
  Bounds:
    lower bound: 3.17e-8
    upper bound: 1.0e-2
  gp seed: 42
  standardize response: false
  Trend:
    estimate trend: true
  Options:
    max degree: 2
    reduced basis: false
    p-norm: 1.0
    scaler type: none
    regression solver type: SVD
    verbosity: 1
  kernel type: squared exponential
  Length-scale Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  verbosity: 1
```

7.2.67.6.2 use_derivatives

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, its use with Surpack Gaussian process is not recommended.

7.2.67.6.3 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.67.6.3.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.67.6.3.2 header

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.67.6.3.3 eval_id

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.67.6.3.4 interface_id

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent [custom_annotated](#)

7.2.67.6.3.5 annotated

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.6.3.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009      1.1 0.0001996404857  0.2601620081  0.759955
0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.67.6.3.7 active_only

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.67.6.4 export_approx_points_file

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
--	--	--	--------------------------	------------------------------

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.67.6.4.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) annotated.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.6.4.2 header

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)

- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.67.6.4.3 `eval_id`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.67.6.4.4 `interface_id`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [export_approx_points_file](#)

- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.67.6.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.6.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [sbgo](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.7 ego

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)

Use the Efficient Global Optimization method

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		gaussian_process	Gaussian Process surrogate model
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations

Description

In the case of `ego`, the efficient global optimization (EGO) method is used to calculate bounds. By default, the Surfpack GP (Kriging) model is used, but the Dakota implementation may be selected instead. If `use_derivatives` is specified the GP model will be built using available derivative data (Surfpack GP only).

See [efficient_global](#) for more information.

7.2.67.7.1 `gaussian_process`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	GP Implementation (Group 1)	surfpack	Use the Surfpack version of Gaussian Process surrogates

			dakota	Select the built in Gaussian Process surrogate
			experimental	Use the experimental Gaussian Process surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the `dakota` version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.67.7.1.1 surfpack

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	Exports surrogate model in user-specified format(s)

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.67.7.1.2 `export_model`

- [Keywords Area](#)
- [method](#)

- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)

- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.2.67.7.1.3 filename_prefix

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: `exported_surrogate`

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.2.67.7.1.4 formats

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format

	Optional	binary_archive	Surrogate model binary archive file format
--	-----------------	--------------------------------	--

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

7.2.67.7.1.5 `text_archive`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.67.7.1.6 binary_archive

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.67.7.1.7 dakota

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the `Surfpack` GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.67.7.1.8 experimental

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)

Use the experimental Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		options_file	Filename for a YAML file that specifies Gaussian process options

Description

The experimental GP surrogate is an emerging capability intended to supersede the `dakota` and `surfpack` GP implementations by unifying the best features of both. It also includes additional features such as the ability to export the model and then use it from Python.

See Also

These keywords may also be of interest:

- [experimental_gaussian_process](#)

7.2.67.7.1.9 export_model

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.2.67.7.1.10 `filename_prefix`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)

- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.2.67.7.1.11 formats

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)

- [experimental](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See [export_model](#) and the entries for the format selection keywords for further information.

7.2.67.7.1.12 [text_archive](#)

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.67.7.1.13 `binary_archive`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.67.7.1.14 options_file

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [gaussian_process](#)
- [experimental](#)
- [options_file](#)

Filename for a YAML file that specifies Gaussian process options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the Gaussian process that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: gp_options.yaml

GP Parameters:
  Sigma Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  scaler name: standardization
  num restarts: 20
  Nugget:
    fixed nugget: 0.0
    estimate nugget: true
  Bounds:
    lower bound: 3.17e-8
```

```
    upper bound: 1.0e-2
gp seed: 42
standardize response: false
Trend:
  estimate trend: true
Options:
  max degree: 2
  reduced basis: false
  p-norm: 1.0
  scaler type: none
  regression solver type: SVD
  verbosity: 1
kernel type: squared exponential
Length-scale Bounds:
  lower bound: 1.0e-2
  upper bound: 1.0e2
verbosity: 1
```

7.2.67.7.2 use_derivatives

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.67.7.3 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.67.7.3.1 custom_annotated

- [Keywords Area](#)
- [method](#)

- [global_interval_est](#)
- [ego](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.7.3.2 header

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.67.7.3.3 eval_id

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)

- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.67.7.3.4 interface_id

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.67.7.3.5 annotated

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [import_build_points_file](#)

- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.7.3.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.7.3.7 active_only

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.67.7.4 export_approx_points_file

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)

- [ego](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: export_points_file

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.67.7.4.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:


```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1      x2      obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9      1.1      0.0002      0.26      0.76
2             0.90009  1.1 0.0001996404857  0.2601620081  0.759955
3             0.89991  1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.67.7.4.2 header

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.67.7.4.3 eval_id

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.67.7.4.4 interface_id

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.67.7.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009       1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991       1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.7.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ego](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.67.8 ea

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [ea](#)

Use an evolutionary algorithm

Specification

Alias: none

Argument(s): none

Description

In this approach, the evolutionary algorithm from Coliny, [coliny_ea](#), is used to perform the interval optimization with no surrogate model involved. Again, this option of `ea` can support interval optimization over discrete variables.

7.2.67.9 lhs

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.67.10 rng

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (`mt19937`)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.67.10.1 mt19937

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
```

```
samples = 10
seed = 98765
rng mt19937
```

7.2.67.10.2 rnum2

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.67.11 model_pointer

- [Keywords Area](#)
- [method](#)
- [global_interval_est](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.68 bayes_calibration

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

Bayesian calibration

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)
- [package_queso](#)

Specification

Alias: nond_bayes_calibration

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Bayesian Calibration Method (Group 1)	queso	Markov Chain Monte Carlo algorithms from the QUESO package
			gpmsa	(Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Bayesian calibration
			wasabi	(Experimental Method) Non-MCMC Bayesian inference using interval analysis

		dream	DREAM (DiffeRential Evolution Adaptive Metropolis)
		muq	Markov Chain Monte Carlo algorithms from the MUQ package
	Optional	experimental_- design	(Experimental) Adaptively select experimental designs for iterative Bayesian updating
	Optional	calibrate_error_- multipliers	Calibrate hyper-parameter multipliers on the observation error covariance
	Optional	burn_in_samples	Manually specify the burn in period for the MCMC chain.
	Optional	posterior_stats	Compute information- theoretic metrics on posterior parameter distribution
	Optional	chain_diagnostics	Compute diagnostic metrics for Markov chain
	Optional	model_evidence	Calculate model evidence (marginal likelihood of model) when using Bayesian methods
	Optional	model_discrepancy	(Experimental) Post-calibration calculation of model discrepancy correction

	Optional	sub_sampling_period	Specify a sub-sampling of the MCMC chain
	Optional	probability_levels	Specify probability levels at which to compute credible and prediction intervals
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	model_pointer	Identifier for model block to be used by a method
	Optional	scaling	Turn on scaling for variables, responses, and constraints

Description

Bayesian calibration methods take prior information on parameter values (in the form of prior distributions) and observational data (e.g. from experiments) and infer posterior distributions on the parameter values. When the computational simulation is then executed with samples from the posterior parameter distributions, the results that are produced are consistent with ("agree with") the experimental data. Calibrating parameters from a computational simulation model requires a likelihood function that specifies the likelihood of observing a particular observation given the model and its associated parameterization; Dakota assumes a Gaussian likelihood function. The algorithms that produce the posterior distributions on model parameters are most commonly Monte Carlo Markov Chain (MCMC) sampling algorithms. MCMC methods require many samples, often tens of thousands, so in the case of model calibration, often emulators of the computational simulation are used. For more details on the algorithms underlying the methods, see the Dakota User's manual.

Dakota has four classes of Bayesian calibration methods: QUESO/DRAM, GPMSA, DREAM, and WASABI.

1. The QUESO methods use components from the QUESO library (Quantification of Uncertainty for Estimation, Simulation, and Optimization) developed at The University of Texas at Austin. Dakota uses its DRAM (Delayed Rejected Adaptive Metropolis) algorithm, and variants, for the MCMC sampling.
2. GPMSA (Gaussian Process Models for Simulation Analysis) is an approach developed at Los Alamos National Laboratory and Dakota currently uses the QUESO implementation. It constructs Gaussian process models to emulate the expensive computational simulation as well as model discrepancy. GPMSA also has extensive features for calibration, such as the capability to include a model discrepancy term and the capability to model functional data such as time series data. This is an experimental capability and not all features are available in Dakota yet.
3. DREAM (DiffeRential Evolution Adaptive Metropolis) is a method that runs multiple different chains simul-

taneously for global exploration, and automatically tunes the proposal covariance during the process by a self-adaptive randomized subspace sampling. [89].

4. WASABI: Non-MCMC Bayesian inference via interval analysis

Usage Tips

The Bayesian capabilities are under active development. At this stage, the QUESO methods in Dakota are the most advanced and robust, followed by DREAM, followed by GPMSA and WASABI which are not yet ready for production use.

The prior distribution is characterized by the properties of the uncertain variables. Correlated priors are only supported for unbounded normal, untruncated lognormal, uniform, exponential, gumbel, frechet, and weibull distributions and require specification of `standardized_space`, for example, for QUESO [standardized_space](#)

Note that as of Dakota 6.2, the field responses and associated field data may be used with QUESO and DREAM. That is, the user can specify field simulation data and field experiment data, and Dakota will interpolate and provide the proper residuals to the Bayesian calibration.

7.2.68.1 queso

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

Markov Chain Monte Carlo algorithms from the QUESO package

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)
- [package_queso](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		chain_samples	Number of Markov Chain Monte Carlo posterior samples

	Optional		seed	Seed of the random number generator
	Optional		rng	Selection of a random number generator
	Optional		emulator	Use an emulator or surrogate model to evaluate the likelihood function
	Optional		standardized_ - space	Perform Bayesian inference in standardized probability space
	Optional		logit_transform	Utilize the logit transformation to reduce sample rejection for bounded domains
	Optional		export_chain_ - points_file	Export the MCMC chain to the specified filename
	Optional		dram	Use the DRAM MCMC algorithm
	Optional (<i>Choose One</i>)	MCMC Algorithm (Group 1)	delayed_rejection	Use the Delayed Rejection MCMC algorithm
			adaptive_ - metropolis	Use the Adaptive Metropolis MCMC algorithm
			metropolis_ - hastings	Use the Metropolis-Hastings MCMC algorithm
			multilevel	Use the multilevel MCMC algorithm.
	Optional		pre_solve	Perform deterministic optimization for MAP before Bayesian calibration

	Optional	proposal_covariance	Defines the technique used to generate the MCMC proposal covariance.
	Optional	options_file	File containing advanced QUESO options

Description

The `queso` method supports the following MCMC algorithms: DRAM (Delayed Rejection Adaptive Metropolis), delayed rejection (DR) only, adaptive metropolis (AM) only, pure Metropolis Hasting (MH)s, and multilevel (ML).

When calibrating fast-running simulation models, use of an emulator is not typically warranted. For slower models, using an emulator model in the MCMC sampling will greatly improve the speed, since the Monte Carlo Markov Chain will generate thousands of samples on the emulator instead of the real simulation code. An emulator may be specified with the keyword `emulator`, followed by a `gaussian_process` emulator, a `pce` emulator (polynomial chaos expansion), or a `sc` emulator (stochastic collocation). For the `gaussian_process` emulator, the user must specify whether to use the `surfpack` or `dakota` version of the Gaussian process. The user can define the number of samples `build_samples` from which the emulator should be built. It is also possible to build the Gaussian process from points read in from the `import_points_file` and to export approximation-based sample evaluations using `export_points_file`. For `pce` or `sc`, the user can define a `sparse_grid_level`.

There are a variety of ways the user can specify the proposal covariance matrix which is very important in governing the samples generated in the chain. The proposal covariance specifies the covariance structure of a multivariate normal distribution. The user can specify `proposal_covariance`, followed by `derivatives`, `prior`, `values`, or `filename`. The derivative specification involves forming the Hessian of the misfit function (the negative log likelihood). When derivative information is available inexpensively (e.g. from an emulator), the derived-based proposal covariance forms a more accurate proposal distribution, resulting in lower rejection rates and faster chain mixing. The prior setting involves constructing the proposal from the variance of the prior distributions of the parameters being calibrated. When specifying the proposal covariance with values or from a file, the user can choose to specify only the diagonals of the covariance matrix with `diagonal` or to specify the full covariance matrix with `matrix`.

There are two other controls for QUESO. The `pre_solve` option enables the user to start the chain at an optimal point, the Maximum A Posteriori (MAP) point. This is the point in parameter space that maximizes the log posterior, (defined as the log-likelihood minus the log prior). A deterministic optimization method is used to obtain the MAP point, and the MCMC chain is then started at the best point found in the optimization. The second factor is a `logit_transform`, which performs an internal variable transformation from bounded domains to unbounded domains in order to reduce sample rejection due to an out-of-bounds condition.

Note that as of Dakota 6.2, the field data capability may be used with QUESO. That is, the user can specify field simulation data and field experiment data, and Dakota will interpolate and provide the proper residuals to the Bayesian calibration.

7.2.68.1.1 chain_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [chain_samples](#)

Number of Markov Chain Monte Carlo posterior samples

Specification

Alias: samples

Argument(s): INTEGER

Default: method-dependent

Description

The `chain_samples` keyword indicates the number of draws from the posterior distribution to perform. When an emulator is active, this will be the number of samples on the constructed surrogate model.

Default Behavior

The default number of chain samples is method-dependent. QUESO and MUQ methods use 1000. DREAM uses (number of generations) x (number of chains), resulting in `chain_samples` close to that specified.

Usage Tips

MCMC methods typically require a large number of chain samples to converge, often thousands to millions.

Examples

```
method
  bayes_calibration queso
  chain_samples = 20000
```

7.2.68.1.2 seed

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.68.1.3 rng

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.68.1.3.1 mt19937

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.68.1.3.2 rnum2

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.68.1.4 emulator

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

Use an emulator or surrogate model to evaluate the likelihood function

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required (Choose One)	Emulator Type (Group 1)	gaussian_process	Gaussian Process surrogate model
			pce	Polynomial Chaos Expansion surrogate model
			ml_pce	Multilevel Polynomial Chaos Expansion as an emulator model.
			mf_pce	Multifidelity Polynomial Chaos Expansion as an emulator model.
			sc	Stochastic Collocation as an emulator model.
			mf_sc	Multifidelity Stochastic Collocation as an emulator model.

Description

This keyword describes the type of emulator used when calculating the likelihood function for the Bayesian calibration. The emulator can be a Gaussian process, polynomial chaos expansion, or stochastic collocation.

7.2.68.1.4.1 [gaussian_process](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	GP Implementation (Group 1)	surfpack	Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate
	Optional		build_samples	Number of initial model evaluations used in build phase
	Optional		posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.68.1.4.2 surfpack

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.68.1.4.3 dakota

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.68.1.4.4 build_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [build_samples](#)

Number of initial model evaluations used in build phase

Specification

Alias: none

Argument(s): INTEGER

Description

The number of build points or training points used in the initial phase of an algorithm or model construction. Typically these are the initial set of data points used to construct (train) a surrogate model (emulator). If the number of `build_samples` is less than the minimum number of points required to build the surrogate, Dakota will augment the samples to obtain the minimum required.

Examples

Perform GP-based adaptive importance sampling, building the GP with 100 points and then performing 100 approximate evaluations to evaluate the probability.

```
method
  gpais
    build_samples = 100
    samples_on_emulator = 100
    max_iterations = 5
    response_levels = -1.065
```

7.2.68.1.4.5 posterior_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```

bayes_calibration queso
  chain_samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack build_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives

```

7.2.68.1.4.6 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.1.4.7 custom_annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	header	Enable header row in custom-annotated tabular file
	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1         0.0002  0.26            0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.1.4.8 header

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.1.4.9 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.1.4.10 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.68.1.4.11 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.12 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [gaussian_process](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.13 `active_only`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.1.4.14 `pce`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)

Polynomial Chaos Expansion surrogate model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional	Group 1	p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Required (<i>Choose One</i>)		quadrature_order	Order for tensor-products of Gaussian quadrature rules
			sparse_grid_level	Level to use in sparse grid integration or interpolation
			cubature_integrand	Cubature using Stroud rules and their extensions
			expansion_order	The (initial) order of a polynomial expansion
			orthogonal_least_ - interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion- _file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_ covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a polynomial chaos expansion (PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)

7.2.68.1.4.15 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_ adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either uniform or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.1.4.16 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.1.4.17 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.1.4.18 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.19 decay

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.20 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.21 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [queso](#)
- [emulator](#)
- [pce](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive generalized  
  p_refinement  
    max_refinement_iterations = 20  
    convergence_tol = 1.e-4  
    sparse_grid_level = 1
```

7.2.68.1.4.22 quadrature_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_ - preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`, where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and `method-multifidelity_stoch_collocation-quadrature_order_sequence`

7.2.68.1.4.23 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.24 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.25 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.26 sparse_grid_level

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.68.1.4.27 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.28 `restricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.29 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed

sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.30 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.31 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.32 cubature_integrand

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [cubature_integrand](#)

Cubature using Stroud rules and their extensions

Specification

Alias: none

Argument(s): INTEGER

Description

Multi-dimensional integration by Stroud cubature rules [80] and extensions [94], as specified with `cubature_integrand`. A total-order expansion is used, where the isotropic order p of the expansion is half of the integrand order, rounded down. The total number of terms N for an isotropic total-order expansion of order p over n variables is given by

$$N = 1 + P = 1 + \sum_{s=1}^p \frac{1}{s!} \prod_{r=0}^{s-1} (n+r) = \frac{(n+p)!}{n!p!}$$

Since the maximum integrand order is currently five for normal and uniform and two for all other types, at most second- and first-order expansions, respectively, will be used. As a result, cubature is primarily useful for global sensitivity analysis, where the Sobol' indices will provide main effects and, at most, two-way interactions. In addition, the random variable set must be independent and identically distributed (*iid*), so the use of `askey` or `wiener` transformations may be required to create *iid* variable sets in the transformed space (as well as to allow usage of the higher order cubature rules for normal and uniform). Note that global sensitivity analysis often assumes uniform bounded regions, rather than precise probability distributions, so the *iid* restriction would not be problematic in that case.

7.2.68.1.4.33 expansion_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)

The (initial) order of a polynomial expansion

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.
	Required (<i>Choose One</i>)	Group 1	collocation_points	Number of collocation points used to estimate expansion coefficients
			collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
			expansion_samples	Number of simulation samples used to estimate the expected value of a set of PCE coefficients
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions

Description

When the `expansion_order` for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product

quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with `expansion_samples`). In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

Linear regression (specified with either `collocation_points` or `collocation_ratio`). A total-order expansion is used and must be specified using `expansion_order` as described in the previous option. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the `tensor_grid` keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If `collocation_points` or `collocation_ratio` is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the `noise_tolerance`, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders ($1, \dots, \text{expansion_order}$) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

7.2.68.1.4.34 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.35 `basis_type`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.

			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.
--	--	--	-------------------------	---

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.1.4.36 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.1.4.37 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.1.4.38 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.39 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.40 `soft_convergence_limit`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.41 `collocation_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (Choose One)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
			basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
			basis_pursuit_denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
			least_angle_regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
			least_absolute_shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.

	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_ - iterations	Maximum iterations in determining polynomial coefficients

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_ - points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation-

_ratio-collocation_points_sequence

7.2.68.1.4.42 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group LSQ Regression Approach (Group 1)	Dakota Keyword svd	Dakota Keyword Description Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.1.4.43 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.1.4.44 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.1.4.45 `orthogonal_matching_pursuit`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.1.4.46 `noise_tolerance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.47 [basis_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.1.4.48 `basis_pursuit_denoising`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: `bpdn`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.1.4.49 `noise_tolerance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)

- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.50 `least_angle_regression`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.1.4.51 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.52 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors. The l_2 penalty used when performing compressed sensing with elastic net.
	Optional		l2_penalty	

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.1.4.53 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.54 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.1.4.55 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.1.4.56 [noise_only](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.1.4.57 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.1.4.58 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.1.4.59 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.1.4.60 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.68.1.4.61 tensor_grid

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.1.4.62 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.63 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.1.4.64 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
			basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
			basis_pursuit_denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

		least_angle_regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

	Optional	max_solver_ - iterations	Maximum iterations in determining polynomial coefficients
--	-----------------	--	---

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p, the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.1.4.65 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.
--	--	--	---------------------------------------	--

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.1.4.66 `svd`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.1.4.67 `equality_constrained`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.1.4.68 [orthogonal_matching_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.1.4.69 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.70 basis_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.1.4.71 `basis_pursuit_denoising`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.1.4.72 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.73 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.1.4.74 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.75 [least_absolute_shrinkage](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.1.4.76 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.77 `l2_penalty`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.1.4.78 `cross_validation`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)

- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.1.4.79 [noise_only](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.1.4.80 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.1.4.81 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.1.4.82 `response_scaling`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.1.4.83 `use_derivatives`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.1.4.84 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.1.4.85 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.86 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.1.4.87 expansion_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [expansion_samples](#)

Number of simulation samples used to estimate the expected value of a set of PCE coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Description

In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

A corresponding sequence specification is documented at, e.g., [expansion_samples_sequence](#) and [expansion_samples_sequence](#)

See Also

These keywords may also be of interest:

- [expansion_samples_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-expansion-_samples_sequence

7.2.68.1.4.88 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [expansion_samples](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.89 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.1.4.90 custom_annotated

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3             0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.68.1.4.91 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.1.4.92 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.1.4.93 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.94 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.95 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.96 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.1.4.97 `posterior_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```
bayes_calibration queso
  chain_samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack build_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives
```

7.2.68.1.4.98 orthogonal_least_interpolation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: least_interpolation oli

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points	Number of collocation points used to estimate expansion coefficients
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models. File containing points you wish to use to build a surrogate
	Optional	import_build_points_file	
	Optional	posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.1.4.99 `collocation_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation_ratio-collocation_points_sequence

7.2.68.1.4.100 tensor_grid

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.1.4.101 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.102 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

	Optional	active_only	Import only active variables from tabular data file
--	-----------------	-----------------------------	---

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.1.4.103 custom_annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```

%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9        1.1         0.0002         0.26           0.76
2             0.90009    1.1 0.0001996404857  0.2601620081   0.759955
3             0.89991    1.1 0.0002003604863  0.2598380081   0.760045
...

```

7.2.68.1.4.104 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.1.4.105 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.106 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.107 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)

- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.108 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.109 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.1.4.110 posterior_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```
bayes_calibration queso
chain_samples = 2000 seed = 348
delayed_rejection
emulator
  gaussian_process surfpack build_samples = 30
  posterior_adaptive max_iterations = 10
  proposal_covariance derivatives
```

7.2.68.1.4.111 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.1.4.112 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.1.4.113 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.1.4.114 export_expansion_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.1.4.115 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.1.4.116 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.1.4.117 ml_pce

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)

Multilevel Polynomial Chaos Expansion as an emulator model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		allocation_control	Sample allocation approach for multilevel expansions
	Optional		discrepancy_emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)	Group 1	expansion_order_sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

	Optional <i>(Choose One)</i>	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional <i>(Choose One)</i>	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multilevel polynomial chaos expansion (ML PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using ML PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multilevel_polynomial_chaos](#)

7.2.68.1.4.118 [allocation_control](#)

- [Keywords Area](#)

- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)

Sample allocation approach for multilevel expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Multilevel Sample Allocation Control (Group 1)	Dakota Keyword estimator_variance	Dakota Keyword Description Variance of mean estimator within multilevel polynomial chaos
			rip_sampling	Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Description

Multilevel expansions, including regression-based polynomial chaos expansion (PCE) and function train (FT) expansions, require a sample allocation strategy. Three options are currently available:

- allocation based on assuming a convergence rate for the estimator variance (for regression PCE)
- restricted isometry property (RIP) sampling (for regression PCE via compressed sensing)
- rank sampling (for FT)

Default Behavior

Current defaults, when `allocation_control` is not specified, are estimator variance for PCE and rank sampling for FT.

7.2.68.1.4.119 `estimator_variance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [estimator_variance](#)

Variance of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		estimator_rate	Rate of convergence of mean estimator within multilevel polynomial chaos

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ , with default values that may be overridden as part of this specification block.

This approach is supported for regression-based PCE approaches (over-determined least squares, under-determined compressed sensing, or orthogonal least interpolation).

In practice, it can be challenging to estimate a smooth convergence rate for estimator variance in the presence of abrupt transitions in the quality of sparse recoveries. As a result, sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the convergence of the estimator variance, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```

method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
  orthogonal_matching_pursuit
  expansion_order_sequence = 2
  pilot_samples = 10
  collocation_ratio = .9
  allocation_control
  estimator_variance estimator_rate = 2.5
  seed = 1237
  convergence_tolerance = .01

```

7.2.68.1.4.120 estimator_rate

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [estimator_variance](#)
- [estimator_rate](#)

Rate of convergence of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): REAL

Default: 2

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ .

The default values are $\gamma = 1$ and $\kappa = 2$ (adopts a more aggressive sample profile by assuming a faster convergence rate than Monte Carlo). This advanced specification option allows to user to specify κ , overriding the default.

7.2.68.1.4.121 rip_sampling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [rip_sampling](#)

Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multilevel polynomial chaos with compressed sensing may allocate the number of samples per level based on the restricted isometry property (RIP), applied to recovery at level l :

$$N_l \geq s_l \log^3(s_l) L_l \log(C_l)$$

for sparsity s , cardinality C , and mutual coherence L . The adaptive algorithm starts from a pilot sample, shapes the profile based on observed sparsity, and iterates until convergence. In practice, RIP sampling levels are quite conservative, and a collocation ratio constraint ($N_l \leq rC_l$, where r defaults to 2) must be enforced on the profile.

The algorithm relies on observed sparsity, it is appropriate for use with regularized solvers for compressed sensing. It employs orthogonal matching pursuit (OMP) by default and automatically activates cross-validation in order to choose the best noise parameter value for the recovery.

This capability is **experimental**. Sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the number of sparse coefficient sets recovered for each level, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```
method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
    orthogonal_matching_pursuit
    expansion_order_sequence = 2
    pilot_samples = 10
    collocation_ratio = .9
    allocation_control rip_sampling
    seed = 1237
    convergence_tolerance = .01
```

7.2.68.1.4.122 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword distinct	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,  
  multilevel_polynomial_chaos  
  expansion_order_sequence = 2  
  collocation_ratio = .9  
  orthogonal_matching_pursuit  
  discrepancy_emulation recursive
```

7.2.68.1.4.123 distinct

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.1.4.124 recursive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.1.4.125 `expansion_order_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)

	Optional	Group 1	basis_type collocation_ratio	<p>Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.</p> <p>Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.</p>
			expansion_ - samples_sequence	<p>Sequence of expansion samples used in a multi-stage polynomial chaos expansion</p> <p>Each level entry of the <code>expansion_ - samples_ - sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.</p> <p>A corresponding scalar specification is documented at, e.g., expansion_ - samples</p>

	Optional	import_build_points_file	File containing points you wish to use to build a surrogate
--	-----------------	--	---

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.68.1.4.126 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.127 `basis_type`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.1.4.128 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.1.4.129 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.1.4.130 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.131 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.132 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.133 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares

		orthogonal_-matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit_-denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle_-regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_-shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.1.4.134 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.1.4.135 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.1.4.136 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.1.4.137 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.1.4.138 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.1.4.139 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.140 [basis_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.1.4.141 [basis_pursuit_denoising](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.1.4.142 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.143 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.1.4.144 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.145 `least_absolute_shrinkage`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: `lasso`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.1.4.146 `noise_tolerance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.147 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.1.4.148 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.1.4.149 noise_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.1.4.150 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.1.4.151 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.1.4.152 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.1.4.153 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.1.4.154 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.1.4.155 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.156 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.1.4.157 `expansion_samples_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.68.1.4.158 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.159 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification**Alias:** import_points_file**Argument(s):** STRING**Default:** no point import from a file**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.1.4.160 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)

- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002      0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.161 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.1.4.162 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.1.4.163 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.164 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.165 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.166 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.1.4.167 `orthogonal_least_interpolation`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: `least_interpolation` `oli`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.1.4.168 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.1.4.169 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.1.4.170 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.171 `import_build_points_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional <i>(Choose One)</i>	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.1.4.172 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.1.4.173 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.1.4.174 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.175 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.176 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26         0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.177 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.178 `active_only`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.1.4.179 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.1.4.180 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.1.4.181 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.1.4.182 `export_expansion_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.1.4.183 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.1.4.184 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [ml_pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.1.4.185 `mf_pce`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)

Multifidelity Polynomial Chaos Expansion as an emulator model.

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional	Group 1	p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_ - emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)		quadrature_order_ - sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_ - sequence	Sequence of sparse grid levels used in a multi-stage expansion
			expansion_order_ - sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_ - interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
Dakota Version 6.16 Reference Manual generated on May 12, 2022				

	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multifidelity polynomial chaos expansion (MF PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using MF PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)

7.2.68.1.4.186 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.1.4.187 `uniform`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [mf_pce](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.1.4.188 `dimension_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.1.4.189 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.190 decay

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.191 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.192 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive generalized  
  p_refinement  
    max_refinement_iterations = 20  
    convergence_tol = 1.e-4  
    sparse_grid_level = 1
```

7.2.68.1.4.193 allocation_control

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity polynomial chaos

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.68.1.4.194 [greedy](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multifidelity polynomial chaos supports greedy refinement strategies, spanning regression and projection approaches for computing expansion coefficients. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within a unified competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any $z/p/\beta/\beta^*$ level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity regression starts from a zeroth-order reference expansion (a constant) for each level, and generates candidate refinements for each level that are competed in an integrated greedy competition. The number of new samples for the incremented candidate expansion order is determined from the collocation ratio. In this case, the number of candidates for each level is limited to one uniform refinement of the current expansion order.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement uniform
  expansion_order_sequence = 0
  collocation_ratio = .9 seed = 160415
  orthogonal_matching_pursuit
  convergence_tolerance 1.e-2
```

The next example employs generalized sparse grids within a greedy multifidelity competition. Each modeling level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all model levels is competed within an integrated greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each model level and the number of candidates grows rapidly with random dimension and grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement dimension_adaptive generalized
  sparse_grid_level_sequence = 0 unrestricted
  convergence_tolerance 1.e-8
```

7.2.68.1.4.195 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Discrepancy Emulation Approach (Group 1)	distinct	Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.68.1.4.196 `distinct`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.1.4.197 `recursive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.1.4.198 quadrature_order_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.68.1.4.199 dimension_preference

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.200 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.201 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.202 sparse_grid_level_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.68.1.4.203 `dimension_preference`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.204 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.205 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.206 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.207 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.208 expansion_order_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.
	Required (<i>Choose One</i>)	Group 1	collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

			expansion_ samples_sequence	<p>Sequence of expansion samples used in a multi-stage polynomial chaos expansion. Each level entry of the <code>expansion_ _samples_ _sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach. A corresponding scalar specification is documented at, e.g., expansion_ _samples</p>
--	--	--	---	---

	Optional	import_build_points_file	File containing points you wish to use to build a surrogate
--	-----------------	--	---

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.68.1.4.209 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.210 basis_type

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.1.4.211 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.1.4.212 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.1.4.213 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.214 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.215 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.1.4.216 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares

		orthogonal_-matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit_-denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle_-regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_-shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.1.4.217 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.1.4.218 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.1.4.219 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.1.4.220 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.1.4.221 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.1.4.222 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.223 [basis_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.1.4.224 [basis_pursuit_denoising](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.1.4.225 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.226 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.1.4.227 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.228 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.1.4.229 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.1.4.230 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.1.4.231 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.1.4.232 noise_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.1.4.233 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.1.4.234 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.1.4.235 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.1.4.236 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.1.4.237 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.1.4.238 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.239 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.1.4.240 `expansion_samples_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.68.1.4.241 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.242 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification**Alias:** import_points_file**Argument(s):** STRING**Default:** no point import from a file**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.1.4.243 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)

- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002      0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.244 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.1.4.245 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.1.4.246 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.247 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.248 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.249 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.1.4.250 `orthogonal_least_interpolation`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: `least_interpolation` `oli`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.1.4.251 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.1.4.252 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.1.4.253 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.1.4.254 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.1.4.255 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.1.4.256 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.1.4.257 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)

- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.258 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.1.4.259 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26          0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.4.260 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...

```

7.2.68.1.4.261 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.1.4.262 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.1.4.263 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.1.4.264 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.1.4.265 `export_expansion_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.1.4.266 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.1.4.267 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.1.4.268 `sc`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)

Stochastic Collocation as an emulator model.

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Required (<i>Choose One</i>)	Interpolation Grid Type (Group 2)	quadrature_order	Order for tensor-products of Gaussian quadrature rules
			sparse_grid_level	Level to use in sparse grid integration or interpolation
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects stochastic collocation (SC) model to use in the Bayesian likelihood calculations. Most specification options are carried over for using SC as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [stoch_collocation](#)

7.2.68.1.4.269 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.

			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
--	--	--	------------------------------------	--

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.1.4.270 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.1.4.271 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.
--	--	--	-----------------------------	---

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.1.4.272 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.273 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.274 h_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [queso](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group h-refinement Type (Group 1)	Dakota Keyword uniform	Dakota Keyword Description Refine an expansion uniformly in all dimensions.
			dimension_-adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either `p_refinement` or `h_refinement`, and either refinement specification can be either `uniform` or `dimension_adaptive`. The `dimension_adaptive` case can be further specified as either `sobol` or `generalized` (decay not supported). Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls. The `h_refinement` specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify `piecewise` in the case of `h_refinement`). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current `uniform` and `dimension_adaptive` options.

7.2.68.1.4.275 `uniform`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.1.4.276 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.1.4.277 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.278 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.279 `local_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.68.1.4.280 `max_refinement_iterations`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.68.1.4.281 quadrature_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`,

where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and [method-multifidelity_stoch_collocation-quadrature_order_sequence](#)

7.2.68.1.4.282 dimension_preference

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.283 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: `non_nested` unless automated refinement; sparse grids: `nested`

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.284 `non_nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.285 `sparse_grid_level`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation

			hierarchical	Employ a hierarchical sparse grid construction
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.68.1.4.286 `dimension_preference`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.287 nodal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.68.1.4.288 hierarchical

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.68.1.4.289 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.290 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed

sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.291 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.292 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.293 `piecewise`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.68.1.4.294 `askey`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.1.4.295 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.1.4.296 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, it's use with Surpack Gaussian process is not recommended.

7.2.68.1.4.297 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.1.4.298 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [sc](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.1.4.299 `mf_sc`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)

Multifidelity Stochastic Collocation as an emulator model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_ - emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)	Interpolation Grid Type (Group 2)	quadrature_order_ - sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_ - sequence	Sequence of sparse grid levels used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multifidelity stochastic collocation (MF SC) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using MF SC as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multifidelity_stoch_collocation](#)

7.2.68.1.4.300 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.1.4.301 `uniform`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)

- [mf_sc](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.1.4.302 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either [sobol](#) or [generalized](#) for stochastic collocation and either [sobol](#), [decay](#), or [generalized](#) for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.68.1.4.303 [sobol](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: [generalized](#)

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.304 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.305 h_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group h-refinement Type (Group 1)	Dakota Keyword uniform	Dakota Keyword Description Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either p_refinement or h_refinement, and either refinement specification can be either uniform or dimension_adaptive. The dimension_adaptive case can be further specified as either sobol or generalized (decay not supported). Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls. The h_refinement specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify piecewise in the case of h_refinement). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current uniform and dimension_adaptive options.

7.2.68.1.4.306 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.1.4.307 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either [sobol](#) or [generalized](#) for stochastic collocation and either [sobol](#), [decay](#), or [generalized](#) for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.68.1.4.308 [sobol](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: [generalized](#)

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.309 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.1.4.310 local_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.68.1.4.311 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.68.1.4.312 allocation_control

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity stochastic collocation

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.68.1.4.313 greedy

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity stochastic collocation

Specification

Alias: none

Argument(s): none

Description

Multifidelity stochastic collocation supports greedy refinement strategies using tensor and sparse grids for both nodal and hierarchical collocation approaches. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within an integrated competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity stochastic collocation using nodal interpolation starts from a zeroth-order expansion (a constant) for each level, and generates uniform candidate refinements for each level that are competed in a greedy competition. The number of new samples for the incremented candidate expansion order is determined from the quadrature rules of the new sparse grid level. In this case, the number of candidates for each level is limited to one uniform refinement of the current sparse grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
    nodal
    allocation_control greedy
    p_refinement uniform
    sparse_grid_level_sequence = 0 unrestricted
    convergence_tolerance 1.e-3
```

The next example employs generalized sparse grids and hierarchical interpolation. Each level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all levels is competed within a unified greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each level and the number of candidates grows rapidly with random dimension and grid level.

```

method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
  hierarchical
  allocation_control greedy
  p_refinement dimension_adaptive generalized
  sparse_grid_level_sequence = 0 unrestricted
  convergence_tolerance 1.e-8

```

7.2.68.1.4.314 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword distinct	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.68.1.4.315 `distinct`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.1.4.316 `recursive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.1.4.317 `quadrature_order_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		dimension_ - preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.68.1.4.318 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.319 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: `non_nested` unless automated refinement; sparse grids: `nested`

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.320 `non_nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.321 `sparse_grid_level_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation

			hierarchical	Employ a hierarchical sparse grid construction
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.68.1.4.322 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.1.4.323 nodal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.68.1.4.324 hierarchical

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.68.1.4.325 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.326 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.1.4.327 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.1.4.328 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.1.4.329 `piecewise`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.68.1.4.330 `askey`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.1.4.331 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.1.4.332 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, it's use with Surpack Gaussian process is not recommended.

7.2.68.1.4.333 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.1.4.334 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [emulator](#)
- [mf_sc](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.1.5 `standardized_space`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [standardized_space](#)

Perform Bayesian inference in standardized probability space

Specification

Alias: none

Argument(s): none

Description

This option transforms the inference process (MCMC sampling and any emulator model management) into a standardized probability space.

The variable transformations performed are as described in [askey](#).

Default Behavior

The default for the Gaussian process and no emulator options is to perform inference in the original probability space (no transformation). Polynomial chaos and stochastic collocation emulators, on the other hand, are always formed in standardized probability space, such that the inference process is also performed in this standardized space.

Expected Output

The user will see the truth model evaluations performed in the original space, whereas any method diagnostics relating to the MCMC samples (e.g., QUESO data in the outputData directory) will report points and response data (response gradients and Hessians, if present, will differ but response values will not) that correspond to the transformed space.

Usage Tips

Selecting `standardized_space` generally has the effect of scaling the random variables to be of comparable magnitude, which can improve the efficiency of the Bayesian inference process.

Examples

```
method,
    bayes_calibration queso
    samples = 2000 seed = 348
    dram
    standardized_space
```

7.2.68.1.6 logit_transform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [logit_transform](#)

Utilize the logit transformation to reduce sample rejection for bounded domains

Specification

Alias: none

Argument(s): none

Description

The logit transformation performs an internal variable transformation from bounded domains to unbounded domains in order to reduce sample rejection due to an out-of-bounds condition.

Default Behavior

This option is experimental at present, and is therefore defaulted off.

Usage Tips

This option can be helpful when regions of high posterior density exist in the corners of a multi-dimensional bounded domain. In these cases, it may be difficult to generate feasible samples from the proposal density, such that transformation to unbounded domains may greatly reduce sample rejection rates.

Examples

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  dram
  logit_transform
```

7.2.68.1.7 export_chain_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [export_chain_points_file](#)

Export the MCMC chain to the specified filename

Specification

Alias: none

Argument(s): STRING

Default: chain export to default filename

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

The filename to which the final MCMC posterior chain will be exported.

Default Behavior No export to file.

Expected Output

A tabular data file will be produced in the specified format (annotated by default) containing samples from the posterior distribution.

Usage Tips

Additional Discussion

7.2.68.1.7.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [export_chain_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and `(fully) annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857  0.2601620081      0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081      0.760045
...
```

7.2.68.1.7.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.1.7.3 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.1.7.4 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.1.7.5 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [export_chain_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.7.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [export_chain_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.1.8 dram

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [dram](#)

Use the DRAM MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

The type of Markov Chain Monte Carlo used. This keyword specifies the use of DRAM, (Delayed Rejection Adaptive Metropolis) [40].

Default Behavior

Five MCMC algorithm variants are supported: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Usage Tips

If the user knows very little about the proposal covariance, using `dram` is a recommended strategy. The proposal covariance is adaptively updated, and the delayed rejection may help improve low acceptance rates.

Examples

```
method,
  bayes_calibration queso
  dram
  samples = 10000 seed = 348
```

7.2.68.1.9 `delayed_rejection`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [delayed_rejection](#)

Use the Delayed Rejection MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: `dram`

Description

This keyword specifies the use of the Delayed Rejection algorithm in which there can be a delay in rejecting samples from the chain. That is, the "DR" part of DRAM is used but the "AM" part is not, rather a regular Metropolis-Hastings algorithm is used.

Default Behavior

Five MCMC algorithm variants are supported: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Usage Tips

If the user knows something about the proposal covariance or the proposal covariance is informed through derivative information, using `delayed_rejection` is preferred over `dram`: the proposal covariance is already being informed by derivative information and the adaptive metropolis is not necessary.

Examples

```
method,
  bayes_calibration queso
  delayed_rejection
  samples = 10000 seed = 348
```

See Also

These keywords may also be of interest:

- [proposal_covariance](#)

7.2.68.1.10 adaptive_metropolis

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [adaptive_metropolis](#)

Use the Adaptive Metropolis MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of the Adaptive Metropolis algorithm. That is, the "AM" part of DRAM is used but the "DR" part is not: specifying this keyword activates only the Adaptive Metropolis part of the MCMC algorithm, in which the covariance of the proposal density is updated adaptively.

Default Behavior

Five MCMC algorithm variants are supported in QUESO: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Four MCMC variants are currently supported in MUQ: `dram`, `delayed_rejection`, `metropolis_hastings` and `adaptive_metropolis`.

Usage Tips

If the user knows very little about the proposal covariance, but doesn't want to incur the cost of using full dram with both delayed rejection and adaptive metropolis, specifying only `adaptive_metropolis` offers a good strategy.

Examples

```
method,
  bayes_calibration queso
  adaptive_metropolis
  samples = 10000 seed = 348
```

7.2.68.1.11 metropolis_hastings

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [metropolis_hastings](#)

Use the Metropolis-Hastings MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of a Metropolis-Hastings algorithm for the MCMC chain generation. This means there is no delayed rejection and no adaptive proposal covariance updating as in DRAM.

Default Behavior

Five MCMC algorithm variants are supported currently in QUESO: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Four MCMC algorithm variants are currently supported in MUQ: `dram`, `delayed_rejection`, `metropolis_hastings` and `adaptive_metropolis`.

Usage Tips

If the user wants to use Metropolis-Hastings, possibly as a comparison to the other methods which involve more chain adaptation, this is the MCMC type to use.

Examples

```
method,
  bayes_calibration queso
  metropolis_hastings
  samples = 10000 seed = 348
```

7.2.68.1.12 multilevel

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [multilevel](#)

Use the multilevel MCMC algorithm.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

Selects the multilevel algorithm described in [73].

Default Behavior

Five MCMC algorithm variants are supported: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Usage Tips

The multilevel algorithm is a more experimental algorithm than the other MCMC approaches mentioned above. It works well in cases where the prior can be "evolved" to a posterior in a structured way. Currently, the multilevel option is not in production form.

Examples

```
method,
  bayes_calibration queso
  multilevel
  samples = 10000 seed = 348
```

7.2.68.1.13 pre_solve

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [pre_solve](#)

Perform deterministic optimization for MAP before Bayesian calibration

Specification

Alias: none

Argument(s): none

Default: nip pre-solve for emulators

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Pre-solve Optimizer (Group 1)	Dakota Keyword	Dakota Keyword Description
			sqp	Uses a sequential quadratic programming method for underlying optimization
			nip	Uses a nonlinear interior point method for underlying optimization
			none	Deactivates MAP pre-solve prior to initiating the MCMC process.

Description

When specified, Dakota will perform a deterministic derivative-based optimization to maximize the log posterior (minimize the negative log posterior = misfit - log_prior + constant normalization factors). The Markov chain in Bayesian calibration will subsequently be started at the best point found in the optimization (the MAP point), which can eliminate the need for "burn in" of the chain in which some initial portion of the chain is discarded. Note that both optimization methods available ([sqp](#) and [nip](#)) require derivatives of the negative log posterior, either first-order in the case of SQP (with quasi-Newton Hessians from secant updates) or second-order in the case of full-Newton NIP (with explicit Hessian use). The derivatives will be computed from the same model used for the MCMC process; e.g. if an emulator is used, the emulator derivatives will be used, otherwise they will be based on the user's model specification for the model.

It is important to clarify that the use of the Hessian of the negative log posterior within a full Newton solver does not strictly require Hessians from the model response quantities of interest (QoIs). Rather, the Hessian of the negative log posterior is formed from an exact Hessian of the negative log prior and a misfit Hessian that can be either exact or approximated: the full misfit Hessian can be formed using QoI residuals, gradients, and Hessians or the

Gauss-Newton approximate misfit Hessian can be formed using only QoI gradients [16]. This Hessian composition is configured automatically based on MAP solver selection and the emulator's or simulation model's support for derivatives.

Default Behavior The default MAP pre-solve behavior depends on the use of an emulator model within the inference process.

If there is an emulator (for which derivatives are easily computed), then the MAP pre-solve is active by default and a full Newton NIP formulation is selected if OPT++ is available. The default use of a MAP pre-solve can be overridden using "pre_solve none" and the default selection of OPT++ full Newton NIP can be replaced with NPSOL SQP using "pre_solve sqp." Depending on the emulator's support for derivatives of the simulated QoI (gradients for dakota GP and stochastic collocation; gradients and Hessians for surpack GP and polynomial chaos), the contribution of the misfit Hessian to the Hessian of the negative log posterior will be computed either using the full misfit Hessian or its Gauss-Newton approximation (refer to Bayesian chapter in [16]).

If no emulator model is specified, then the pre-solve is bypassed by default and the MCMC chain is initiated from user-specified (or default) initial value for the prior distributions of the random variables. This default can be overridden by specifying "pre_solve nip" for a full Newton NIP solution or "pre_solve sqp" for an NPSOL SQP solution. Both MAP pre-solve approaches require QoI gradients from the simulation model, and the full Newton approach can further leverage QoI Hessians when available (though not required due to the Gauss-Newton approximation, as explained previously).

Expected Output When pre-solve is enabled, the output will include a deterministic optimization, followed by a Bayesian calibration. The final results will include the MAP point as well as posterior statistics from the MCMC chain. The MAP point that is reported is the point with highest posterior probability spanning both the pre-solve and the subsequent MCMC chain; it will most commonly reflect the end state of the pre-solve, although it can reflect subsequent improvements from the chain evolution, should they occur.

Examples

```
method
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator
    pce sparse_grid_level = 2
    pre_solve nip # default for emulators
```

7.2.68.1.13.1 sqp

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [pre_solve](#)
- [sqp](#)

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `sqp` keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

7.2.68.1.13.2 `nip`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [pre_solve](#)
- [nip](#)

Uses a nonlinear interior point method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

7.2.68.1.13.3 `none`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [pre_solve](#)
- [none](#)

Deactivates MAP pre-solve prior to initiating the MCMC process.

Specification

Alias: none

Argument(s): none

Description

Pre-solving for the maximum a posteriori probability (MAP) point could be undesirable when testing MCMC performance or too expensive to pursue in some settings. The option "none" provides an override that deactivates the pre-solve option for cases where it would normally be active by default (e.g., for emulator models).

7.2.68.1.14 proposal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)

Defines the technique used to generate the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Proposal Covariance Source (Group 1)	prior	Uses the covariance of the prior distributions to define the MCMC proposal covariance.
			derivatives	Use derivatives to inform the MCMC proposal covariance.
			values	Specifies matrix values to use as the MCMC proposal covariance.

			filename	Uses a file to import a user-specified MCMC proposal covariance.
--	--	--	--------------------------	--

Description

The proposal covariance is used to define a multivariate normal (MVN) jumping distribution used to create new points within a Markov chain. That is, a new point in the chain is determined by sampling within a MVN probability density with prescribed covariance that is centered at the current chain point. The accuracy of the proposal covariance has a significant effect on rejection rates and the efficiency of chain mixing.

Default Behavior

The default proposal covariance is `prior` when no emulator is present; `derivatives` when an emulator is present.

Expected Output

The effect of the proposal covariance is reflected in the MCMC chain values and the rejection rates, which can be seen in the diagnostic outputs from the QUESO solver within the `QuesoDiagnostics` directory.

Usage Tips

When derivative information is available inexpensively (e.g., from an emulator model), the derived-based proposal covariance forms a more accurate proposal distribution, resulting in lower rejection rates and faster chain mixing.

7.2.68.1.14.1 `prior`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [prior](#)

Uses the covariance of the prior distributions to define the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		multiplier	Multiplier to scale prior variance

Description

This keyword selection results in definition of the MCMC proposal covariance from the covariance of the prior distributions. This covariance is currently assumed to be diagonal without correlation.

Default Behavior

This is the default `proposal_covariance` option.

Usage Tips

Since this proposal covariance is defined globally, the chain does not need to be periodically restarted using local updates to this proposal. However, it is usually effective to adapt the proposal using one of the adaptive metropolis MCMC options.

Examples

```
method,
  bayes_calibration queso
    samples = 2000 seed = 348
  dram
  proposal_covariance prior
```

7.2.68.1.14.2 multiplier

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [prior](#)
- [multiplier](#)

Multiplier to scale prior variance

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

The initial proposal covariance will be given by the prior variance times the multiplier.

Default Behavior When using prior-based proposal covariance, the default is to use the prior variance for the proposal (multiplier = 1.0)

Usage Tips

The prior variance may result in too tight or narrow a proposal covariance. The multiplier can be used to scale all entries of the prior variance in determining the initial proposal covariance.

Examples

```
method
  bayes_calibration queso
    samples = 2000 seed = 348
    dram
    proposal_covariance prior
    multiplier 0.1
```

7.2.68.1.14.3 derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [derivatives](#)

Use derivatives to inform the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		update_period	Period at which to update derivative-based proposal covariance

Description

This keyword selection results in definition of the MCMC proposal covariance from the Hessian of the misfit function (negative log likelihood), where this Hessian is defined from either a Gauss-Newton approximation (using only first derivatives of the calibration terms) or a full Hessian (using values, first derivatives, and second derivatives of the calibration terms). If this Hessian is indeterminate, it will be corrected as described in [16]

Default Behavior The default is `prior` based proposal covariance. This is a more advanced option that exploits structure in the form of the likelihood.

Expected Output

When derivatives are specified for defining the proposal covariance, the misfit Hessian and its inverse (the MVN proposal covariance) will be output to the standard output stream.

Usage Tips

The full Hessian of the misfit is used when either supported by the emulator in use (for PCE and surfpack GP, but not SC or dakota GP) or by the user's response specification (Hessian type is not "no_hessians"), in the case of no emulator. When this full Hessian is indefinite and cannot be inverted to form the proposal covariance, fallback to the positive semi-definite Gauss-Newton Hessian is employed.

Since this proposal covariance is locally accurate, it should be updated periodically using the `update_period` option. While the adaptive metropolis option can be used in combination with derivative-based preconditioning, it is generally preferable to instead decrease the proposal update period due to the improved local accuracy of this approach.

Examples

Generate a 2000 sample posterior chain, using derivatives to initialize the proposal covariance at the start of the chain.

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator pce sparse_grid_level = 2
  proposal_covariance derivatives
```

7.2.68.1.14.4 update_period

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [derivatives](#)
- [update_period](#)

Period at which to update derivative-based proposal covariance

Specification

Alias: none

Argument(s): INTEGER

Description

For derivative-based proposal covariance, this specifies the period (number of accepted MCMC samples) after which the proposal covariance is updated using derivative values at the current chain point.

Default Behavior

When `update_period` is not specified, derivatives will inform the proposal covariance at the start of the chain, but not updated further.

Usage Tips

The `update_period` should be tailored to the size of the total chain, accounting for the relative expense of derivative-based proposal updates.

Examples

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator pce sparse_grid_level = 2
  proposal_covariance derivatives
  update_period = 40 # update proposal covariance every 40 points
```

7.2.68.1.14.5 values

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [values](#)

Specifies matrix values to use as the MCMC proposal covariance.

Specification

Alias: none

Argument(s): REALLIST

Child Keywords:

	Required/Optional	Description of Group Values For (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		diagonal	Specifies the diagonal matrix format when specifying a user-specified proposal covariance.

			matrix	Specifies the full matrix format when specifying a user-specified proposal covariance.
--	--	--	------------------------	--

Description

This keyword selection results in definition of the MCMC proposal covariance from user-specified matrix values. The matrix input format must be declared as either a full matrix or a matrix diagonal.

Default Behavior

This option is not the default, and generally implies special a priori knowledge from the user.

Usage Tips

This option is not supported for the case of transformations to standardized probability space.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  values ... # See leaf nodes for required format option
```

7.2.68.1.14.6 diagonal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [values](#)
- [diagonal](#)

Specifies the diagonal matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    diagonal values 1.0e6 1.0e-1
```

7.2.68.1.14.7 matrix

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [values](#)
- [matrix](#)

Specifies the full matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    matrix values 1.0 0.1
                 0.1 2.0
```

7.2.68.1.14.8 filename

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [filename](#)

Uses a file to import a user-specified MCMC proposal covariance.

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Values For (Group 1)	Dakota Keyword diagonal	Dakota Keyword Description Specifies the diagonal matrix format when importing a user-specified proposal covariance.
			matrix	Specifies the full matrix format when importing a user-specified proposal covariance.

Description

This keyword selection results in definition of the MCMC proposal covariance from importing data a user-specified filename. This import must be declared as either a full matrix or a matrix diagonal.

Default Behavior

This option is not the default, and generally implies special a priori knowledge from the user.

Usage Tips

This option is not supported for the case of transformations to standardized probability space.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    filename ... # See leaf nodes for required format option
```

7.2.68.1.14.9 diagonal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [filename](#)
- [diagonal](#)

Specifies the diagonal matrix format when importing a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  diagonal filename 'dakota_cantilever_queso.diag.dat'
```

7.2.68.1.14.10 matrix

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [proposal_covariance](#)
- [filename](#)
- [matrix](#)

Specifies the full matrix format when importing a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  matrix filename 'dakota_cantilever_queso.matrix.dat'
```

7.2.68.1.15 options_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [queso](#)
- [options_file](#)

File containing advanced QUESO options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

Allow power users to override default QUESO options such as hyper-parameter priors, initial chain position, and proposal covariance (including hyper-parameters), which might not be accessible through Dakota user input.

Default Behavior No advanced options used.

Usage Tips The `options_file` offers a final override over other options settings, which are applied with the following precedence:

1. QUESO library defaults
2. Dakota hard-coded defaults
3. Settings from Dakota input file
4. Settings from the advanced `options_file`

Examples

```
# FILE: dakota_queso.opts

# Base QUESO Environment options
env_subDisplayFileName = QuesoDiagnostics/display
env_subDisplayAllowedSet = 0 1
env_displayVerbosity = 2
env_seed = 2460
```

7.2.68.2 gpmsa

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)

(Experimental) Gaussian Process Models for Simulation Analysis (GPMSA) Bayesian calibration

Topics

This keyword is related to the topics:

- [package_queso](#)
- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		chain_samples	Number of Markov Chain Monte Carlo posterior samples
	Optional		seed	Seed of the random number generator
	Optional		rng	Selection of a random number generator
	Required		build_samples	Number of initial model evaluations used in build phase
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		standardized_space	Perform Bayesian inference in standardized probability space
	Optional		logit_transform	Utilize the logit transformation to reduce sample rejection for bounded domains

	Optional		gpmsa_normalize	Enable GPMSA-internal normalization
	Optional		export_chain_points_file	Export the MCMC chain to the specified filename
	Optional (<i>Choose One</i>)	MCMC Algorithm (Group 1)	dram	Use the DRAM MCMC algorithm
			delayed_rejection	Use the Delayed Rejection MCMC algorithm
			adaptive_metropolis	Use the Adaptive Metropolis MCMC algorithm
			metropolis_hastings	Use the Metropolis-Hastings MCMC algorithm
	Optional		proposal_covariance	Defines the technique used to generate the MCMC proposal covariance.
	Optional		options_file	File containing advanced QUESO/GPMSA options

Description

GPMSA (Gaussian Process Models for Simulation Analysis) is a surrogate-based Markov Chain Monte Carlo Bayesian calibration method. Dakota's GPMSA is an experimental capability and not ready for production use at this time.

Central to GPMSA is the construction of a Gaussian Process emulator from simulation runs collected at various settings of input parameters. The emulator is a statistical model of the system response, and it is used to incorporate the observational data to improve system predictions and constrain or calibrate the unknown parameters. The GPMSA code draws heavily on the theory developed in the seminal Bayesian calibration paper by Kennedy and O'Hagan [56]. The particular approach in GPMSA was developed by the Los Alamos National Laboratory statistics group and documented in [49]. Dakota's GPMSA capability comes from the QUESO package developed at UT Austin.

Usage Tips:

Configuring GPMSA essentially involves identifying the simulation build data, the experiment data, the calibration and configuration (state) variables, and any necessary algorithm controls. The GP surrogate model is automatically constructed internal to the algorithm and need not be specified through Dakota input.

Dakota's GPMSA implementation is not intended for production use. There are a number of known limitations, including:

- Only works for scalar and multivariate responses, not field responses. Field responses will be treated as a single multi-variate response set. Consequently, simulation and experiment data must have the same dimensions.

- When build data is not imported a design of experiments will be conducted over all calibration and scenario variables present.
- Experiment data is required (one cannot pose the simulation data as a set of residuals with the assumption of 0-valued experiments).
- Output and diagnostics are limited. Advanced users will need to examine QUESO output files (potentially written in a transformed scaled space) in the QuesoDiagnostic directory

Examples

The following input file fragment illustrates GPMSA-based Bayesian calibration of 3 β variables with a uniform prior, with 3 configuration (scenario) variables x . A total of 60 simulation build points are provided in `sim_data.dat`, which contains columns for each β , followed by each x , and then the simulation response 'lin'. Each row of the experiment data file `y_exp_with_var.dat` contains the values of the 3 x variables, followed by the value of 'lin' and its observation error (variance).

```
method
  bayes_calibration gpmsa
  chain_samples 1000
  seed 2460
  build_samples 60
  import_build_points_file 'sim_data.dat' freeform
  export_chain_points_file 'posterior.dat'
  burn_in_samples = 100 sub_sampling_period = 2
  posterior_stats kl

variables
  uniform_uncertain 3
  upper_bounds 0.4500 -0.1000 0.4000
  initial_point 0.2750 -0.3000 0.1000
  lower_bounds -0.1000 -0.5000 -0.2000
  descriptors 'beta0' 'beta1' 'beta2'

  continuous_state 3
  upper_bounds 3 * 1.0
  initial_state 3 * 0.5
  lower_bounds 3 * 0.0
  descriptors 'x0' 'x1' 'x2'

responses
  descriptors 'lin'
  calibration_terms 1
  calibration_data_file 'y_exp_with_var.dat'
  freeform
  num_experiments 5
  num_config_variables 3
  experiment_variance_type 'scalar'
  no_gradients
  no_hessians
```

7.2.68.2.1 chain_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [chain_samples](#)

Number of Markov Chain Monte Carlo posterior samples

Specification

Alias: samples

Argument(s): INTEGER

Default: method-dependent

Description

The `chain_samples` keyword indicates the number of draws from the posterior distribution to perform. When an emulator is active, this will be the number of samples on the constructed surrogate model.

Default Behavior

The default number of chain samples is method-dependent. QUESO and MUQ methods use 1000. DREAM uses (number of generations) x (number of chains), resulting in `chain_samples` close to that specified.

Usage Tips

MCMC methods typically require a large number of chain samples to converge, often thousands to millions.

Examples

```
method
  bayes_calibration queso
  chain_samples = 20000
```

7.2.68.2.2 seed

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.68.2.3 rng

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (`mt19937`)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.68.2.3.1 mt19937

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.68.2.3.2 rnum2

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)

- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.68.2.4 build_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [build_samples](#)

Number of initial model evaluations used in build phase

Specification

Alias: none

Argument(s): INTEGER

Description

The number of build points or training points used in the initial phase of an algorithm or model construction. Typically these are the initial set of data points used to construct (train) a surrogate model (emulator). If the number of `build_samples` is less than the minimum number of points required to build the surrogate, Dakota will augment the samples to obtain the minimum required.

Examples

Perform GP-based adaptive importance sampling, building the GP with 100 points and then performing 100 approximate evaluations to evaluate the probability.

```
method
  gpais
    build_samples = 100
    samples_on_emulator = 100
    max_iterations = 5
    response_levels = -1.065
```

7.2.68.2.5 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.2.5.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.2.5.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)

- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.2.5.3 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.2.5.4 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [import_build_points_file](#)

- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.2.5.5 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.2.5.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.68.2.6 standardized_space

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [standardized_space](#)

Perform Bayesian inference in standardized probability space

Specification

Alias: none

Argument(s): none

Description

This option transforms the inference process (MCMC sampling and any emulator model management) into a standardized probability space.

The variable transformations performed are as described in [askey](#).

Default Behavior

The default for the Gaussian process and no emulator options is to perform inference in the original probability space (no transformation). Polynomial chaos and stochastic collocation emulators, on the other hand, are always formed in standardized probability space, such that the inference process is also performed in this standardized space.

Expected Output

The user will see the truth model evaluations performed in the original space, whereas any method diagnostics relating to the MCMC samples (e.g., QUESO data in the outputData directory) will report points and response data (response gradients and Hessians, if present, will differ but response values will not) that correspond to the transformed space.

Usage Tips

Selecting `standardized_space` generally has the effect of scaling the random variables to be of comparable magnitude, which can improve the efficiency of the Bayesian inference process.

Examples

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  dram
  standardized_space
```

7.2.68.2.7 logit_transform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [logit_transform](#)

Utilize the logit transformation to reduce sample rejection for bounded domains

Specification

Alias: none

Argument(s): none

Description

The logit transformation performs an internal variable transformation from bounded domains to unbounded domains in order to reduce sample rejection due to an out-of-bounds condition.

Default Behavior

This option is experimental at present, and is therefore defaulted off.

Usage Tips

This option can be helpful when regions of high posterior density exist in the corners of a multi-dimensional bounded domain. In these cases, it may be difficult to generate feasible samples from the proposal density, such that transformation to unbounded domains may greatly reduce sample rejection rates.

Examples

```
method,  
  bayes_calibration queso  
  samples = 2000 seed = 348  
  dram  
  logit_transform
```

7.2.68.2.8 gpmsa_normalize

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [gpmsa_normalize](#)

Enable GPMSA-internal normalization

Specification

Alias: none

Argument(s): none

Description

Normalize variables and data (surrogate model build and experiment) when constructing the underlying Gaussian Process emulator. When specified, this option will normalize calibration parameters and configuration (scenario) variables based on the min/max of the build data. It will normalize simulation and experiment data by the mean and variance of the simulation build data.

Default Behavior No GPMSA-specific scaling used.

Examples

```
method  
  bayes_calibration gpmsa  
  chain_samples 1000  
  seed 2460  
  ...  
  gpmsa_normalize
```

7.2.68.2.9 export_chain_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)

- [export_chain_points_file](#)

Export the MCMC chain to the specified filename

Specification

Alias: none

Argument(s): STRING

Default: chain export to default filename

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

The filename to which the final MCMC posterior chain will be exported.

Default Behavior No export to file.

Expected Output

A tabular data file will be produced in the specified format (annotated by default) containing samples from the posterior distribution.

Usage Tips

Additional Discussion

7.2.68.2.9.1 [custom_annotated](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [export_chain_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

%eval_id	x1	x2	obj_fn	nln_ineq_con_1	nln_ineq_con_2
1	0.9	1.1	0.0002	0.26	0.76
2	0.90009	1.1	0.0001996404857	0.2601620081	0.759955
3	0.89991	1.1	0.0002003604863	0.2598380081	0.760045
...					

7.2.68.2.9.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.2.9.3 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.2.9.4 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent [custom_annotated](#)

7.2.68.2.9.5 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [export_chain_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.2.9.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [export_chain_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.2.10 dram

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [dram](#)

Use the DRAM MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

The type of Markov Chain Monte Carlo used. This keyword specifies the use of DRAM, (Delayed Rejection Adaptive Metropolis) [40].

Default Behavior

Five MCMC algorithm variants are supported: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Usage Tips

If the user knows very little about the proposal covariance, using `dram` is a recommended strategy. The proposal covariance is adaptively updated, and the delayed rejection may help improve low acceptance rates.

Examples

```
method,
  bayes_calibration queso
  dram
  samples = 10000 seed = 348
```

7.2.68.2.11 `delayed_rejection`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [delayed_rejection](#)

Use the Delayed Rejection MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of the Delayed Rejection algorithm in which there can be a delay in rejecting samples from the chain. That is, the "DR" part of DRAM is used but the "AM" part is not, rather a regular Metropolis-Hastings algorithm is used.

Default Behavior

Five MCMC algorithm variants are supported: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Usage Tips

If the user knows something about the proposal covariance or the proposal covariance is informed through derivative information, using `delayed_rejection` is preferred over `dram`: the proposal covariance is already being informed by derivative information and the adaptive metropolis is not necessary.

Examples

```
method,
  bayes_calibration queso
  delayed_rejection
  samples = 10000 seed = 348
```

See Also

These keywords may also be of interest:

- [proposal_covariance](#)

7.2.68.2.12 adaptive_metropolis

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [adaptive_metropolis](#)

Use the Adaptive Metropolis MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of the Adaptive Metropolis algorithm. That is, the "AM" part of DRAM is used but the "DR" part is not: specifying this keyword activates only the Adaptive Metropolis part of the MCMC algorithm, in which the covariance of the proposal density is updated adaptively.

Default Behavior

Five MCMC algorithm variants are supported in QUESO: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Four MCMC variants are currently supported in MUQ: `dram`, `delayed_rejection`, `metropolis_hastings` and `adaptive_metropolis`.

Usage Tips

If the user knows very little about the proposal covariance, but doesn't want to incur the cost of using full `dram` with both delayed rejection and adaptive metropolis, specifying only `adaptive_metropolis` offers a good strategy.

Examples

```
method,
  bayes_calibration queso
  adaptive_metropolis
  samples = 10000 seed = 348
```

7.2.68.2.13 metropolis_hastings

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [metropolis_hastings](#)

Use the Metropolis-Hastings MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of a Metropolis-Hastings algorithm for the MCMC chain generation. This means there is no delayed rejection and no adaptive proposal covariance updating as in DRAM.

Default Behavior

Five MCMC algorithm variants are supported currently in QUESO: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Four MCMC algorithm variants are currently supported in MUQ: `dram`, `delayed_rejection`, `metropolis_hastings` and `adaptive_metropolis`.

Usage Tips

If the user wants to use Metropolis-Hastings, possibly as a comparison to the other methods which involve more chain adaptation, this is the MCMC type to use.

Examples

```
method,
  bayes_calibration queso
  metropolis_hastings
  samples = 10000 seed = 348
```

7.2.68.2.14 proposal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)

Defines the technique used to generate the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Proposal Covariance Source (Group 1)	prior	Uses the covariance of the prior distributions to define the MCMC proposal covariance.
			derivatives	Use derivatives to inform the MCMC proposal covariance.
			values	Specifies matrix values to use as the MCMC proposal covariance.
			filename	Uses a file to import a user-specified MCMC proposal covariance.

Description

The proposal covariance is used to define a multivariate normal (MVN) jumping distribution used to create new points within a Markov chain. That is, a new point in the chain is determined by sampling within a MVN probability density with prescribed covariance that is centered at the current chain point. The accuracy of the proposal covariance has a significant effect on rejection rates and the efficiency of chain mixing.

Default Behavior

The default proposal covariance is `prior` when no emulator is present; `derivatives` when an emulator is present.

Expected Output

The effect of the proposal covariance is reflected in the MCMC chain values and the rejection rates, which can be seen in the diagnostic outputs from the QUESO solver within the `QuesoDiagnostics` directory.

Usage Tips

When derivative information is available inexpensively (e.g., from an emulator model), the derived-based proposal covariance forms a more accurate proposal distribution, resulting in lower rejection rates and faster chain mixing.

7.2.68.2.14.1 `prior`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [prior](#)

Uses the covariance of the prior distributions to define the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		multiplier	Multiplier to scale prior variance

Description

This keyword selection results in definition of the MCMC proposal covariance from the covariance of the prior distributions. This covariance is currently assumed to be diagonal without correlation.

Default Behavior

This is the default `proposal_covariance` option.

Usage Tips

Since this proposal covariance is defined globally, the chain does not need to be periodically restarted using local updates to this proposal. However, it is usually effective to adapt the proposal using one of the adaptive metropolis MCMC options.

Examples

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  dram
  proposal_covariance prior
```

7.2.68.2.14.2 multiplier

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [prior](#)
- [multiplier](#)

Multiplier to scale prior variance

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

The initial proposal covariance will be given by the prior variance times the multiplier.

Default Behavior When using prior-based proposal covariance, the default is to use the prior variance for the proposal (multiplier = 1.0)

Usage Tips

The prior variance may result in too tight or narrow a proposal covariance. The multiplier can be used to scale all entries of the prior variance in determining the initial proposal covariance.

Examples

```
method
  bayes_calibration queso
    samples = 2000 seed = 348
  dram
    proposal_covariance prior
    multiplier 0.1
```

7.2.68.2.14.3 derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [derivatives](#)

Use derivatives to inform the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		update_period	Period at which to update derivative-based proposal covariance

Description

This keyword selection results in definition of the MCMC proposal covariance from the Hessian of the misfit function (negative log likelihood), where this Hessian is defined from either a Gauss-Newton approximation (using only first derivatives of the calibration terms) or a full Hessian (using values, first derivatives, and second derivatives of the calibration terms). If this Hessian is indeterminate, it will be corrected as described in [16]

Default Behavior The default is `prior` based proposal covariance. This is a more advanced option that exploits structure in the form of the likelihood.

Expected Output

When derivatives are specified for defining the proposal covariance, the misfit Hessian and its inverse (the MVN proposal covariance) will be output to the standard output stream.

Usage Tips

The full Hessian of the misfit is used when either supported by the emulator in use (for PCE and surpack GP, but not SC or dakota GP) or by the user's response specification (Hessian type is not "no_hessians"), in the case of no emulator. When this full Hessian is indefinite and cannot be inverted to form the proposal covariance, fallback to the positive semi-definite Gauss-Newton Hessian is employed.

Since this proposal covariance is locally accurate, it should be updated periodically using the `update_period` option. While the adaptive metropolis option can be used in combination with derivative-based preconditioning, it is generally preferable to instead decrease the proposal update period due to the improved local accuracy of this approach.

Examples

Generate a 2000 sample posterior chain, using derivatives to initialize the proposal covariance at the start of the chain.

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator pce sparse_grid_level = 2
  proposal_covariance derivatives
```

7.2.68.2.14.4 update_period

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [derivatives](#)
- [update_period](#)

Period at which to update derivative-based proposal covariance

Specification

Alias: none

Argument(s): INTEGER

Description

For derivative-based proposal covariance, this specifies the period (number of accepted MCMC samples) after which the proposal covariance is updated using derivative values at the current chain point.

Default Behavior

When `update_period` is not specified, derivatives will inform the proposal covariance at the start of the chain, but not updated further.

Usage Tips

The `update_period` should be tailored to the size of the total chain, accounting for the relative expense of derivative-based proposal updates.

Examples

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator pce sparse_grid_level = 2
  proposal_covariance derivatives
  update_period = 40 # update proposal covariance every 40 points
```

7.2.68.2.14.5 values

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [values](#)

Specifies matrix values to use as the MCMC proposal covariance.

Specification

Alias: none

Argument(s): REALLIST

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Values For (Group 1)	Dakota Keyword diagonal	Dakota Keyword Description Specifies the diagonal matrix format when specifying a user-specified proposal covariance.
			matrix	Specifies the full matrix format when specifying a user-specified proposal covariance.

Description

This keyword selection results in definition of the MCMC proposal covariance from user-specified matrix values. The matrix input format must be declared as either a full matrix or a matrix diagonal.

Default Behavior

This option is not the default, and generally implies special a priori knowledge from the user.

Usage Tips

This option is not supported for the case of transformations to standardized probability space.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  values ... # See leaf nodes for required format option
```

7.2.68.2.14.6 diagonal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [values](#)
- [diagonal](#)

Specifies the diagonal matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    diagonal values 1.0e6 1.0e-1
```

7.2.68.2.14.7 matrix

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [values](#)
- [matrix](#)

Specifies the full matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    matrix values 1.0 0.1
                 0.1 2.0
```

7.2.68.2.14.8 filename

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [gpmsa](#)
- [proposal_covariance](#)
- [filename](#)

Uses a file to import a user-specified MCMC proposal covariance.

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Values For (Group 1)	Dakota Keyword diagonal	Dakota Keyword Description Specifies the diagonal matrix format when importing a user-specified proposal covariance.
			matrix	Specifies the full matrix format when importing a user-specified proposal covariance.

Description

This keyword selection results in definition of the MCMC proposal covariance from importing data a user-specified filename. This import must be declared as either a full matrix or a matrix diagonal.

Default Behavior

This option is not the default, and generally implies special a priori knowledge from the user.

Usage Tips

This option is not supported for the case of transformations to standardized probability space.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  filename ... # See leaf nodes for required format option
```

7.2.68.2.14.9 diagonal

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [filename](#)
- [diagonal](#)

Specifies the diagonal matrix format when importing a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  diagonal filename 'dakota_cantilever_queso.diag.dat'
```

7.2.68.2.14.10 matrix

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [proposal_covariance](#)
- [filename](#)
- [matrix](#)

Specifies the full matrix format when importing a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  matrix filename 'dakota_cantilever_queso.matrix.dat'
```

7.2.68.2.15 options_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [gpmsa](#)
- [options_file](#)

File containing advanced QUESO/GPMSA options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

Allow power users to override default QUESO/GPMSA options such as hyper-parameter priors, initial chain position, and proposal covariance (including hyper-parameters), which might not be accessible through Dakota user input.

Default Behavior No advanced options used.

Usage Tips The `options_file` offers a final override over other options settings, which are applied with the following precedence:

1. QUESO library defaults
2. Dakota hard-coded defaults
3. Settings from Dakota input file
4. Settings from the advanced `options_file`

Examples

```
# FILE: dakota_gpmsa.opts

# Base QUESO Environment options
env_subDisplayFileName = QuesoDiagnostics/display
env_subDisplayAllowedSet = 0 1
env_displayVerbosity = 2
env_seed = 2460

# GPMSA Options
gpmsa_emulator_precision_shape = 2.0
gpmsa_emulator_precision_scale = 0.2
gpmsa_emulator_correlation_strength_alpha = 1.0
```

```
gpmsa_emulator_correlation_strength_beta = 0.1
gpmsa_discrepancy_precision_shape = 1.0
gpmsa_discrepancy_precision_scale = 1e4
gpmsa_discrepancy_correlation_strength_alpha = 1.0
gpmsa_discrepancy_correlation_strength_beta = 0.1
gpmsa_emulator_data_precision_shape = 3.0
gpmsa_emulator_data_precision_scale = 333.333
```

7.2.68.3 wasabi

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

(Experimental Method) Non-MCMC Bayesian inference using interval analysis

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		pushforward_ - samples	(Experimental Capability) Number of samples of the prior to push forward through the model Description: WASABI requires a forward UQ that maps samples from the prior parameter distribution through the model. The corresponding responses then are used to see their relative likelihood according to the density of the observational data. The pushforward_ - samples refers to the number of samples taken from the prior and used in the forward UQ. It typically should be high, e.g. 10000, of which only a small fraction may be in a high density region of the posterior.
	Optional		seed	Seed of the random number generator
	Optional		emulator	Use an emulator or surrogate model to evaluate the likelihood function
	Optional		standardized_ - space	Perform Bayesian inference in standardized probability space

	<p>Required</p> <p>Optional</p>	<p><code>data_distribution</code></p> <p><code>posterior_samples_ - _import_filename</code></p>	<p>(Experimental Capability) Specify the distribution of the experimental data</p> <p>(Experimental Capability) Filename for samples at which the user would like the posterior density calculated</p>
	<p>Optional</p>	<p><code>generate_ - posterior_samples</code></p>	<p>(Experimental Capability) Generate random samples from the posterior density</p> <p>Description: This keyword will result in samples from the prior that have a high posterior density being printed to a file called 'psamples.txt' as a default, or to a file specified by the <code>posterior_ - samples_ - export_ - filename</code>.</p>

	Optional	evaluate_posterior_density	(Experimental Capability) Evaluate the posterior density and output to the specified file Description: This keyword will allow the evaluation of the posterior density for all of the prior samples, typically specified with the number given in <code>pushforward_samples</code> . The density will be printed either to a file named 'pdens.txt' as a default, or to the file specified in <code>posterior_density_export_filename</code> .
--	-----------------	--	--

Description

Offers an alternative to Markov Chain Monte Carlo-based Bayesian inference. This is a nascent capability, not yet ready for production use.

Usage Guidelines: The WASABI method requires an emulator model.

Attention: While the `emulator` specification for WASABI includes the keyword `posterior_adaptive`, it is not yet operational.

Examples

```
method
  bayes_calibration
  wasabi
```

7.2.68.3.1 pushforward_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [pushforward_samples](#)

(Experimental Capability) Number of samples of the prior to push forward through the model

Description: WASABI requires a forward UQ that maps samples from the prior parameter distribution through the model. The corresponding responses then are used to see their relative likelihood according to the density of the

observational data. The `pushforward_samples` refers to the number of samples taken from the prior and used in the forward UQ. It typically should be high, e.g. 10000, of which only a small fraction may be in a high density region of the posterior.

Specification

Alias: none

Argument(s): INTEGER

Default: method-dependent

7.2.68.3.2 `seed`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.68.3.3 emulator

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

Use an emulator or surrogate model to evaluate the likelihood function

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Emulator Type (Group 1)	gaussian_process	Gaussian Process surrogate model
			pce	Polynomial Chaos Expansion surrogate model
			ml_pce	Multilevel Polynomial Chaos Expansion as an emulator model.
			mf_pce	Multifidelity Polynomial Chaos Expansion as an emulator model.
			sc	Stochastic Collocation as an emulator model.
			mf_sc	Multifidelity Stochastic Collocation as an emulator model.

Description

This keyword describes the type of emulator used when calculating the likelihood function for the Bayesian calibration. The emulator can be a Gaussian process, polynomial chaos expansion, or stochastic collocation.

7.2.68.3.3.1 gaussian_process

- [Keywords Area](#)

- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group GP Implementation (Group 1)	Dakota Keyword surfpack	Dakota Keyword Description Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate
	Optional		build_samples	Number of initial model evaluations used in build phase
	Optional		posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the `dakota` version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.68.3.3.2 surfpack

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.68.3.3.3 dakota

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the Surfpack GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to

select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.68.3.3.4 `build_samples`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [build_samples](#)

Number of initial model evaluations used in build phase

Specification

Alias: none

Argument(s): INTEGER

Description

The number of build points or training points used in the initial phase of an algorithm or model construction. Typically these are the initial set of data points used to construct (train) a surrogate model (emulator). If the number of `build_samples` is less than the minimum number of points required to build the surrogate, Dakota will augment the samples to obtain the minimum required.

Examples

Perform GP-based adaptive importance sampling, building the GP with 100 points and then performing 100 approximate evaluations to evaluate the probability.

```
method
  gpais
    build_samples = 100
    samples_on_emulator = 100
    max_iterations = 5
    response_levels = -1.065
```

7.2.68.3.3.5 `posterior_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [gaussian_process](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```
bayes_calibration queso
  chain_samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack build_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives
```

7.2.68.3.3.6 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.3.3.7 custom_annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.3.3.8 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.3.3.9 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.10 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.11 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002         0.26         0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.12 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...

```

7.2.68.3.3.13 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.3.3.14 pce

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)

Polynomial Chaos Expansion surrogate model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Required (<i>Choose One</i>)	Group 1	quadrature_order	Order for tensor-products of Gaussian quadrature rules
			sparse_grid_level	Level to use in sparse grid integration or interpolation

			cubature_integrand	Cubature using Stroud rules and their extensions
			expansion_order	The (initial) order of a polynomial expansion
			orthogonal_least_interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a polynomial chaos expansion (PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)

7.2.68.3.3.15 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.3.3.16 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.3.3.17 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.
--	--	--	-----------------------------	---

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.3.3.18 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.19 decay

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.20 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,  
  polynomial_chaos  
  sparse_grid_level = 3  
  dimension_adaptive p_refinement generalized  
  max_iterations = 20  
  convergence_tol = 1.e-4
```

7.2.68.3.3.21 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
    dimension_adaptive generalized
  p_refinement
    max_refinement_iterations = 20
    convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.68.3.3.22 quadrature_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for

gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`, where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and `method-multifidelity_stoch_collocation-quadrature_order_sequence`

7.2.68.3.3.23 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.24 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: `non_nested` unless automated refinement; sparse grids: `nested`

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.25 `non_nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.26 sparse_grid_level

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (Choose One)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.68.3.3.27 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.28 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it

cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.29 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- `unrestricted`

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.30 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)

- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.31 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.32 cubature_integrand

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [pce](#)
- [cubature_integrand](#)

Cubature using Stroud rules and their extensions

Specification

Alias: none

Argument(s): INTEGER

Description

Multi-dimensional integration by Stroud cubature rules [80] and extensions [94], as specified with `cubature_integrand`. A total-order expansion is used, where the isotropic order p of the expansion is half of the integrand order, rounded down. The total number of terms N for an isotropic total-order expansion of order p over n variables is given by

$$N = 1 + P = 1 + \sum_{s=1}^p \frac{1}{s!} \prod_{r=0}^{s-1} (n+r) = \frac{(n+p)!}{n!p!}$$

Since the maximum integrand order is currently five for normal and uniform and two for all other types, at most second- and first-order expansions, respectively, will be used. As a result, cubature is primarily useful for global sensitivity analysis, where the Sobol' indices will provide main effects and, at most, two-way interactions. In addition, the random variable set must be independent and identically distributed (*iid*), so the use of `askey` or `wiener` transformations may be required to create *iid* variable sets in the transformed space (as well as to allow usage of the higher order cubature rules for normal and uniform). Note that global sensitivity analysis often assumes uniform bounded regions, rather than precise probability distributions, so the *iid* restriction would not be problematic in that case.

7.2.68.3.3.33 expansion_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)

The (initial) order of a polynomial expansion

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		dimension_ - preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.
	Required (Choose One)	Group 1	collocation_points	Number of collocation points used to estimate expansion coefficients
			collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
			expansion_ - samples	Number of simulation samples used to estimate the expected value of a set of PCE coefficients
	Optional		import_build_ - points_file	File containing points you wish to use to build a surrogate
	Optional		posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions

Description

When the `expansion_order` for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with `expansion_samples`). In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

Linear regression (specified with either `collocation_points` or `collocation_ratio`). A total-order expansion is used and must be specified using `expansion_order` as described in the previous option. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear

relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the `tensor_grid` keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If `collocation_points` or `collocation_ratio` is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the `noise_tolerance`, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders (1,...,`expansion_order`) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

7.2.68.3.3.34 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.35 `basis_type`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.3.3.36 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.3.3.37 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.3.3.38 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.39 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.40 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.41 collocation_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit - denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle - regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute - shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	use_sub_samples	Use sub-samples tensor-product quadrature points to build a polynomial chaos

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation_ratio-collocation_points_sequence

7.2.68.3.3.42 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.3.3.43 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.3.3.44 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.3.3.45 `orthogonal_matching_pursuit`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.3.3.46 `noise_tolerance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.47 `basis_pursuit`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.3.3.48 basis_pursuit_denoising

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.3.3.49 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)

- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.50 `least_angle_regression`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.3.3.51 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.52 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors. The l_2 penalty used when performing compressed sensing with elastic net.
	Optional		l2_penalty	

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.3.3.53 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.54 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.3.3.55 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.3.3.56 [noise_only](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.3.3.57 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.3.3.58 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.3.3.59 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.3.3.60 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.68.3.3.61 tensor_grid

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.3.3.62 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.63 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.3.3.64 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
			basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
			basis_pursuit_denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

		least_angle_regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

	Optional	max_solver_ - iterations	Maximum iterations in determining polynomial coefficients
--	-----------------	--	---

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p, the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.3.3.65 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.
--	--	--	---------------------------------------	--

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.3.3.66 `svd`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.3.3.67 `equality_constrained`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.3.3.68 [orthogonal_matching_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.3.3.69 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.70 basis_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.3.3.71 `basis_pursuit_denoising`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.3.3.72 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.73 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.3.3.74 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.75 [least_absolute_shrinkage](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.3.3.76 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.77 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.3.3.78 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)

- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.3.3.79 [noise_only](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.3.3.80 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.3.3.81 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.3.3.82 `response_scaling`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.3.3.83 `use_derivatives`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.3.3.84 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.3.3.85 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.86 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.3.3.87 expansion_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [expansion_samples](#)

Number of simulation samples used to estimate the expected value of a set of PCE coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Description

In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

A corresponding sequence specification is documented at, e.g., [expansion_samples_sequence](#) and [expansion_samples_sequence](#)

See Also

These keywords may also be of interest:

- [expansion_samples_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-expansion-_samples_sequence

7.2.68.3.3.88 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [expansion_samples](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.89 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.3.3.90 custom_annotated

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2      obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1      0.0002      0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.91 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.3.3.92 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.93 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.94 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.95 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.96 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.3.3.97 `posterior_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```
bayes_calibration queso
  chain_samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack build_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives
```

7.2.68.3.3.98 orthogonal_least_interpolation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: least_interpolation oli

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points	Number of collocation points used to estimate expansion coefficients
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models. File containing points you wish to use to build a surrogate
	Optional	import_build_points_file	
	Optional	posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.3.3.99 collocation_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation_ratio-collocation_points_sequence

7.2.68.3.3.100 tensor_grid

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.3.3.101 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.102 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

	Optional	active_only	Import only active variables from tabular data file
--	-----------------	-----------------------------	---

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.3.3.103 custom_annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.68.3.3.104 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.3.3.105 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.106 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.107 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)

- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.108 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.109 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.3.3.110 posterior_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```
bayes_calibration queso
  chain_samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack build_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives
```

7.2.68.3.3.111 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.3.3.112 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.3.3.113 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.3.3.114 export_expansion_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.3.3.115 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.3.3.116 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.3.3.117 ml_pce

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)

Multilevel Polynomial Chaos Expansion as an emulator model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		allocation_control	Sample allocation approach for multilevel expansions
	Optional		discrepancy_emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)	Group 1	expansion_order_sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

	Optional (Choose One)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional (Choose One)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multilevel polynomial chaos expansion (ML PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using ML PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multilevel_polynomial_chaos](#)

7.2.68.3.3.118 [allocation_control](#)

- [Keywords Area](#)

- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)

Sample allocation approach for multilevel expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Multilevel Sample Allocation Control (Group 1)	Dakota Keyword estimator_variance	Dakota Keyword Description Variance of mean estimator within multilevel polynomial chaos
			rip_sampling	Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Description

Multilevel expansions, including regression-based polynomial chaos expansion (PCE) and function train (FT) expansions, require a sample allocation strategy. Three options are currently available:

- allocation based on assuming a convergence rate for the estimator variance (for regression PCE)
- restricted isometry property (RIP) sampling (for regression PCE via compressed sensing)
- rank sampling (for FT)

Default Behavior

Current defaults, when `allocation_control` is not specified, are estimator variance for PCE and rank sampling for FT.

7.2.68.3.3.119 `estimator_variance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [estimator_variance](#)

Variance of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		estimator_rate	Rate of convergence of mean estimator within multilevel polynomial chaos

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ , with default values that may be overridden as part of this specification block.

This approach is supported for regression-based PCE approaches (over-determined least squares, under-determined compressed sensing, or orthogonal least interpolation).

In practice, it can be challenging to estimate a smooth convergence rate for estimator variance in the presence of abrupt transitions in the quality of sparse recoveries. As a result, sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the convergence of the estimator variance, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```

method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
  orthogonal_matching_pursuit
  expansion_order_sequence = 2
  pilot_samples = 10
  collocation_ratio = .9
  allocation_control
  estimator_variance estimator_rate = 2.5
  seed = 1237
  convergence_tolerance = .01

```

7.2.68.3.3.120 estimator_rate

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [estimator_variance](#)
- [estimator_rate](#)

Rate of convergence of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): REAL

Default: 2

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ .

The default values are $\gamma = 1$ and $\kappa = 2$ (adopts a more aggressive sample profile by assuming a faster convergence rate than Monte Carlo). This advanced specification option allows to user to specify κ , overriding the default.

7.2.68.3.3.121 rip_sampling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [rip_sampling](#)

Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multilevel polynomial chaos with compressed sensing may allocate the number of samples per level based on the restricted isometry property (RIP), applied to recovery at level l :

$$N_l \geq s_l \log^3(s_l) L_l \log(C_l)$$

for sparsity s , cardinality C , and mutual coherence L . The adaptive algorithm starts from a pilot sample, shapes the profile based on observed sparsity, and iterates until convergence. In practice, RIP sampling levels are quite conservative, and a collocation ratio constraint ($N_l \leq rC_l$, where r defaults to 2) must be enforced on the profile.

The algorithm relies on observed sparsity, it is appropriate for use with regularized solvers for compressed sensing. It employs orthogonal matching pursuit (OMP) by default and automatically activates cross-validation in order to choose the best noise parameter value for the recovery.

This capability is **experimental**. Sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the number of sparse coefficient sets recovered for each level, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```
method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
    orthogonal_matching_pursuit
    expansion_order_sequence = 2
    pilot_samples = 10
    collocation_ratio = .9
    allocation_control rip_sampling
    seed = 1237
    convergence_tolerance = .01
```

7.2.68.3.3.122 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword distinct	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,  
  multilevel_polynomial_chaos  
  expansion_order_sequence = 2  
  collocation_ratio = .9  
  orthogonal_matching_pursuit  
  discrepancy_emulation recursive
```

7.2.68.3.3.123 distinct

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.3.3.124 recursive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.3.3.125 `expansion_order_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)

	Optional	Group 1	basis_type collocation_ratio	<p>Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.</p> <p>Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.</p>
			expansion_ - samples_sequence	<p>Sequence of expansion samples used in a multi-stage polynomial chaos expansion</p> <p>Each level entry of the <code>expansion_ - samples_ - sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.</p> <p>A corresponding scalar specification is documented at, e.g., expansion_ - samples</p>

	Optional	import_build_points_file	File containing points you wish to use to build a surrogate
--	-----------------	--	---

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.68.3.3.126 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.127 `basis_type`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.3.3.128 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.3.3.129 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.3.3.130 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.131 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.132 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.133 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares

		orthogonal_-matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit_-denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle_-regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_-shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_ - iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.3.3.134 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.3.3.135 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.3.3.136 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.3.3.137 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.3.3.138 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.3.3.139 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.140 [basis_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.3.3.141 [basis_pursuit_denoising](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.3.3.142 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.143 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.3.3.144 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.145 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.3.3.146 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.147 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.3.3.148 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.3.3.149 noise_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.3.3.150 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.3.3.151 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.3.3.152 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.3.3.153 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.3.3.154 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.3.3.155 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.156 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.3.3.157 `expansion_samples_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.68.3.3.158 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.159 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification**Alias:** import_points_file**Argument(s):** STRING**Default:** no point import from a file**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.3.3.160 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)

- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002      0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.161 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.3.3.162 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.3.3.163 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.164 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.165 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.166 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.3.3.167 `orthogonal_least_interpolation`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: `least_interpolation oli`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.3.3.168 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.3.3.169 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.3.3.170 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.171 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.3.3.172 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.3.3.173 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.3.3.174 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.175 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.176 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26         0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.177 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.3.3.178 `active_only`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.3.3.179 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.3.3.180 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.3.3.181 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.3.3.182 `export_expansion_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.3.3.183 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.3.3.184 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [ml_pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.3.3.185 `mf_pce`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)

Multifidelity Polynomial Chaos Expansion as an emulator model.

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional	Group 1	p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_ - emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)		quadrature_order_ - sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_ - sequence	Sequence of sparse grid levels used in a multi-stage expansion
			expansion_order_ - sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_ - interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
Dakota Version 6.16 Reference Manual generated on May 12, 2022				

	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multifidelity polynomial chaos expansion (MF PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using MF PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)

7.2.68.3.3.186 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.3.3.187 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [mf_pce](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed.

7.2.68.3.3.188 `dimension_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.3.3.189 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,  
  polynomial_chaos  
    sparse_grid_level = 3  
    dimension_adaptive p_refinement sobol  
      max_iterations = 20  
      convergence_tol = 1.e-4
```

7.2.68.3.3.190 decay

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.191 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.192 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive generalized  
  p_refinement  
    max_refinement_iterations = 20  
    convergence_tol = 1.e-4  
    sparse_grid_level = 1
```

7.2.68.3.3.193 allocation_control

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity polynomial chaos

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.68.3.3.194 [greedy](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multifidelity polynomial chaos supports greedy refinement strategies, spanning regression and projection approaches for computing expansion coefficients. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within a unified competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any $z/p/\beta/\beta^*$ level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity regression starts from a zeroth-order reference expansion (a constant) for each level, and generates candidate refinements for each level that are competed in an integrated greedy competition. The number of new samples for the incremented candidate expansion order is determined from the collocation ratio. In this case, the number of candidates for each level is limited to one uniform refinement of the current expansion order.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement uniform
  expansion_order_sequence = 0
  collocation_ratio = .9 seed = 160415
  orthogonal_matching_pursuit
  convergence_tolerance 1.e-2
```

The next example employs generalized sparse grids within a greedy multifidelity competition. Each modeling level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all model levels is competed within an integrated greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each model level and the number of candidates grows rapidly with random dimension and grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement dimension_adaptive generalized
  sparse_grid_level_sequence = 0 unrestricted
  convergence_tolerance 1.e-8
```

7.2.68.3.3.195 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Discrepancy Emulation Approach (Group 1)	distinct	Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.68.3.3.196 `distinct`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.3.3.197 recursive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.3.3.198 quadrature_order_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.68.3.3.199 dimension_preference

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.200 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.201 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.202 sparse_grid_level_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.68.3.3.203 `dimension_preference`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.204 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.205 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.206 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.207 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.208 expansion_order_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.
	Required (<i>Choose One</i>)	Group 1	collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

			expansion_ samples_sequence	<p>Sequence of expansion samples used in a multi-stage polynomial chaos expansion. Each level entry of the <code>expansion_ _samples_ _sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach. A corresponding scalar specification is documented at, e.g., expansion_ _samples</p>
--	--	--	---	---

	Optional	import_build_points_file	File containing points you wish to use to build a surrogate
--	-----------------	--	---

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.68.3.3.209 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.210 basis_type

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.3.3.211 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.3.3.212 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.3.3.213 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.214 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.215 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.3.3.216 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares

		orthogonal_-matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit_-denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle_-regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_-shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.3.3.217 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.3.3.218 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.3.3.219 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.3.3.220 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.3.3.221 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.3.3.222 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.223 [basis_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.3.3.224 [basis_pursuit_denoising](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.3.3.225 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.226 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.3.3.227 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.228 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.3.3.229 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.3.3.230 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.3.3.231 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.3.3.232 noise_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.3.3.233 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.3.3.234 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.3.3.235 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.3.3.236 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.68.3.3.237 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.3.3.238 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.239 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.3.3.240 `expansion_samples_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.68.3.3.241 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.242 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification**Alias:** import_points_file**Argument(s):** STRING**Default:** no point import from a file**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.3.3.243 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)

- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1             0.9          1.1          0.0002      0.26          0.76
2             0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3             0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.244 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.3.3.245 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.3.3.246 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.247 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.248 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.249 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.3.3.250 `orthogonal_least_interpolation`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: `least_interpolation oli`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.3.3.251 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.3.3.252 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.3.3.253 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.3.3.254 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional <i>(Choose One)</i>	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.3.3.255 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.3.3.256 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.3.3.257 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)

- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.258 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.3.3.259 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26         0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.3.3.260 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...

```

7.2.68.3.3.261 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.3.3.262 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.3.3.263 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.3.3.264 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.3.3.265 `export_expansion_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.3.3.266 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.3.3.267 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.3.3.268 `sc`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)

Stochastic Collocation as an emulator model.

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Required (<i>Choose One</i>)	Interpolation Grid Type (Group 2)	quadrature_order	Order for tensor-products of Gaussian quadrature rules
			sparse_grid_level	Level to use in sparse grid integration or interpolation
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects stochastic collocation (SC) model to use in the Bayesian likelihood calculations. Most specification options are carried over for using SC as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [stoch_collocation](#)

7.2.68.3.3.269 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.

			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
--	--	--	------------------------------------	--

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.3.3.270 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.3.3.271 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.
--	--	--	-------------	---

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.3.3.272 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.273 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.274 h_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [wasabi](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group h-refinement Type (Group 1)	Dakota Keyword uniform	Dakota Keyword Description Refine an expansion uniformly in all dimensions.
			dimension_-adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either `p_refinement` or `h_refinement`, and either refinement specification can be either `uniform` or `dimension_adaptive`. The `dimension_adaptive` case can be further specified as either `sobol` or `generalized` (decay not supported). Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls. The `h_refinement` specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify `piecewise` in the case of `h_refinement`). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current `uniform` and `dimension_adaptive` options.

7.2.68.3.3.275 `uniform`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.3.3.276 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required (Choose One)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.3.3.277 [sobol](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.278 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.279 local_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.68.3.3.280 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.68.3.3.281 quadrature_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`,

where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and [method-multifidelity_stoch_collocation-quadrature_order_sequence](#)

7.2.68.3.3.282 dimension_preference

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.283 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.284 `non_nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.285 `sparse_grid_level`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation

			hierarchical	Employ a hierarchical sparse grid construction
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.68.3.3.286 `dimension_preference`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.287 nodal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.68.3.3.288 hierarchical

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.68.3.3.289 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.290 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed

sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.291 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.292 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.293 `piecewise`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.68.3.3.294 `askey`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.3.3.295 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.3.3.296 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, it's use with Surpack Gaussian process is not recommended.

7.2.68.3.3.297 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.3.3.298 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [sc](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.3.3.299 `mf_sc`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)

Multifidelity Stochastic Collocation as an emulator model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_ - emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)	Interpolation Grid Type (Group 2)	quadrature_order_ - sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_ - sequence	Sequence of sparse grid levels used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multifidelity stochastic collocation (MF SC) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using MF SC as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multifidelity_stoch_collocation](#)

7.2.68.3.3.300 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.3.3.301 `uniform`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)

- [mf_sc](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.3.3.302 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either [sobol](#) or [generalized](#) for stochastic collocation and either [sobol](#), [decay](#), or [generalized](#) for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.68.3.3.303 [sobol](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: [generalized](#)

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.304 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.305 h_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	h-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either p_refinement or h_refinement, and either refinement specification can be either uniform or dimension_adaptive. The dimension_adaptive case can be further specified as either sobol or generalized (decay not supported). Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls. The h_refinement specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify piecewise in the case of h_refinement). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current uniform and dimension_adaptive options.

7.2.68.3.3.306 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.3.3.307 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either [sobol](#) or [generalized](#) for stochastic collocation and either [sobol](#), [decay](#), or [generalized](#) for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.68.3.3.308 [sobol](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: [generalized](#)

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.309 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.3.3.310 local_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.68.3.3.311 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.68.3.3.312 allocation_control

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity stochastic collocation

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.68.3.3.313 greedy

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity stochastic collocation

Specification

Alias: none

Argument(s): none

Description

Multifidelity stochastic collocation supports greedy refinement strategies using tensor and sparse grids for both nodal and hierarchical collocation approaches. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within an integrated competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity stochastic collocation using nodal interpolation starts from a zeroth-order expansion (a constant) for each level, and generates uniform candidate refinements for each level that are competed in a greedy competition. The number of new samples for the incremented candidate expansion order is determined from the quadrature rules of the new sparse grid level. In this case, the number of candidates for each level is limited to one uniform refinement of the current sparse grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
    nodal
    allocation_control greedy
    p_refinement uniform
    sparse_grid_level_sequence = 0 unrestricted
    convergence_tolerance 1.e-3
```

The next example employs generalized sparse grids and hierarchical interpolation. Each level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all levels is competed within a unified greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each level and the number of candidates grows rapidly with random dimension and grid level.

```

method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
  hierarchical
  allocation_control greedy
  p_refinement dimension_adaptive generalized
  sparse_grid_level_sequence = 0 unrestricted
  convergence_tolerance 1.e-8

```

7.2.68.3.3.314 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword distinct	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.68.3.3.315 `distinct`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.3.3.316 `recursive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.3.3.317 `quadrature_order_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		dimension_ - preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.68.3.3.318 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.319 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: `non_nested` unless automated refinement; sparse grids: `nested`

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.320 `non_nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.321 `sparse_grid_level_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation

			hierarchical	Employ a hierarchical sparse grid construction
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.68.3.3.322 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.3.3.323 nodal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.68.3.3.324 hierarchical

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.68.3.3.325 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.326 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.3.3.327 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.3.3.328 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.3.3.329 `piecewise`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.68.3.3.330 `askey`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.3.3.331 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.3.3.332 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, it's use with Surpack Gaussian process is not recommended.

7.2.68.3.3.333 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.3.3.334 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [emulator](#)
- [mf_sc](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.3.4 `standardized_space`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [standardized_space](#)

Perform Bayesian inference in standardized probability space

Specification

Alias: none

Argument(s): none

Description

This option transforms the inference process (MCMC sampling and any emulator model management) into a standardized probability space.

The variable transformations performed are as described in [askey](#).

Default Behavior

The default for the Gaussian process and no emulator options is to perform inference in the original probability space (no transformation). Polynomial chaos and stochastic collocation emulators, on the other hand, are always formed in standardized probability space, such that the inference process is also performed in this standardized space.

Expected Output

The user will see the truth model evaluations performed in the original space, whereas any method diagnostics relating to the MCMC samples (e.g., QUESO data in the outputData directory) will report points and response data (response gradients and Hessians, if present, will differ but response values will not) that correspond to the transformed space.

Usage Tips

Selecting `standardized_space` generally has the effect of scaling the random variables to be of comparable magnitude, which can improve the efficiency of the Bayesian inference process.

Examples

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  dram
  standardized_space
```

7.2.68.3.5 data_distribution

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [data_distribution](#)

(Experimental Capability) Specify the distribution of the experimental data

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Data Distribution (Group 1)	gaussian	(Experimental Capability) Gaussian error distribution

			obs_data_filename	(Experimental Capability) Filename from which to read experimental data
--	--	--	-----------------------------------	--

7.2.68.3.5.1 gaussian

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [data_distribution](#)
- [gaussian](#)

(Experimental Capability) Gaussian error distribution

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		means	(Experimental Capability) Means of Gaussian error distribution
	Required		covariance	(Experimental Capability) Covariance of a Gaussian error distribution

7.2.68.3.5.2 means

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [data_distribution](#)
- [gaussian](#)
- [means](#)

(Experimental Capability) Means of Gaussian error distribution

Specification

Alias: none

Argument(s): REALLIST

7.2.68.3.5.3 covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [data_distribution](#)
- [gaussian](#)
- [covariance](#)

(Experimental Capability) Covariance of a Gaussian error distribution

Specification

Alias: none

Argument(s): REALLIST

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Values For (Group 1)	Dakota Keyword diagonal	Dakota Keyword Description (Experimental Capability) Diagonal error covariance
			matrix	(Experimental Capability) Symmetric positive definite error covariance

7.2.68.3.5.4 diagonal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [data_distribution](#)
- [gaussian](#)
- [covariance](#)
- [diagonal](#)

(Experimental Capability) Diagonal error covariance

Specification

Alias: none

Argument(s): none

7.2.68.3.5.5 matrix

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [data_distribution](#)
- [gaussian](#)
- [covariance](#)
- [matrix](#)

(Experimental Capability) Symmetric positive definite error covariance

Specification

Alias: none

Argument(s): none

7.2.68.3.5.6 obs_data_filename

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [data_distribution](#)
- [obs_data_filename](#)

(Experimental Capability) Filename from which to read experimental data

Specification

Alias: none

Argument(s): STRING

7.2.68.3.6 posterior_samples_import_filename

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [posterior_samples_import_filename](#)

(Experimental Capability) Filename for samples at which the user would like the posterior density calculated

Specification

Alias: none

Argument(s): STRING

7.2.68.3.7 generate_posterior_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [generate_posterior_samples](#)

(Experimental Capability) Generate random samples from the posterior density

Description: This keyword will result in samples from the prior that have a high posterior density being printed to a file called 'psamples.txt' as a default, or to a file specified by the `posterior_samples_export_filename`.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		posterior_samples-export_filename	(Experimental Capability) Filename for the exported posterior samples

7.2.68.3.7.1 posterior_samples_export_filename

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [generate_posterior_samples](#)
- [posterior_samples_export_filename](#)

(Experimental Capability) Filename for the exported posterior samples

Specification

Alias: none

Argument(s): STRING

7.2.68.3.8 evaluate_posterior_density

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [evaluate_posterior_density](#)

(Experimental Capability) Evaluate the posterior density and output to the specified file

Description: This keyword will allow the evaluation of the posterior density for all of the prior samples, typically specified with the number given in `pushforward_samples`. The density will be printed either to a file named 'pdens.txt' as a default, or to the file specified in `posterior_density_export_filename`.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		posterior_density_export_filename	(Experimental Capability) Filename for the exported posterior density

7.2.68.3.8.1 posterior_density_export_filename

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [wasabi](#)
- [evaluate_posterior_density](#)
- [posterior_density_export_filename](#)

(Experimental Capability) Filename for the exported posterior density

Specification

Alias: none

Argument(s): STRING

7.2.68.4 dream

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [dream](#)

DREAM (DiffeRential Evolution Adaptive Metropolis)

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		chain_samples	Number of Markov Chain Monte Carlo posterior samples
	Optional		seed	Seed of the random number generator
	Optional		chains	Number of chains in DREAM
	Optional		num_cr	Number of candidate points for each crossover.
	Optional		crossover_chain_pairs	Number of chains used in crossover.
	Optional		gr_threshold	Convergence tolerance for the Gelman-Rubin statistic
	Optional		jump_step	Number of generations a long jump step is taken
	Optional		emulator	Use an emulator or surrogate model to evaluate the likelihood function
	Optional		standardized_space	Perform Bayesian inference in standardized probability space

	Optional	export_chain_points_file	Export the MCMC chain to the specified filename
--	-----------------	--	---

Description

The Differential Evolution Adaptive Metropolis algorithm is described in [89]. For the DREAM method, one can define the number of chains used with `chains` (minimum 3). The total number of generations per chain in DREAM is the number of samples (`samples`) divided by the number of chains (`chains`). The number of chains randomly selected to be used in the crossover each time a crossover occurs is `crossover_chain_pairs`. There is an extra adaptation during burn-in, in which DREAM estimates a distribution of crossover probabilities that favors large jumps over smaller ones in each of the chains. Normalization is required to ensure that all of the input dimensions contribute equally. In this process, a discrete number of candidate points for each crossover value is generated. This parameter is `num_cr`. The `gr_threshold` is the convergence tolerance for the Gelman-Rubin statistic which will govern the convergence of the multiple chain process. The integer `jump_step` forces a long jump every `jump_step` generations. For more details about these parameters, see [89].

Attention: While the `emulator` specification for DREAM includes the keyword `posterior_adaptive`, it is not yet operational.

7.2.68.4.1 chain_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [chain_samples](#)

Number of Markov Chain Monte Carlo posterior samples

Specification

Alias: `samples`

Argument(s): INTEGER

Default: method-dependent

Description

The `chain_samples` keyword indicates the number of draws from the posterior distribution to perform. When an emulator is active, this will be the number of samples on the constructed surrogate model.

Default Behavior

The default number of chain samples is method-dependent. QUESO and MUQ methods use 1000. DREAM uses (number of generations) x (number of chains), resulting in `chain_samples` close to that specified.

Usage Tips

MCMC methods typically require a large number of chain samples to converge, often thousands to millions.

Examples

```
method
  bayes_calibration queso
  chain_samples = 20000
```

7.2.68.4.2 seed

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.68.4.3 chains

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [chains](#)

Number of chains in DREAM

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 3

Description

Number of chains in DREAM

7.2.68.4.4 `num_cr`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [num_cr](#)

Number of candidate points for each crossover.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

In DREAM, there is an extra adaptation during burn-in, in which DREAM estimates a distribution of crossover probabilities that favors large jumps over smaller ones in each of the chains. Normalization is required to ensure that all of the input dimensions contribute equally. In this process, a discrete number of candidate points for each crossover value is generated. This parameter is `num_cr`.

7.2.68.4.5 `crossover_chain_pairs`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [crossover_chain_pairs](#)

Number of chains used in crossover.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 3

Description

The number of chains randomly selected to be used in the crossover each time a crossover occurs.

7.2.68.4.6 `gr_threshold`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [gr_threshold](#)

Convergence tolerance for the Gelman-Rubin statistic

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.2

Description

The `gr_threshold` is the convergence tolerance for the Gelman-Rubin statistic which will govern the convergence of the multiple chain process.

7.2.68.4.7 `jump_step`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [jump_step](#)

Number of generations a long jump step is taken

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 5

Description

The integer `jump_step` forces a long jump every `jump_step` generations in DREAM.

7.2.68.4.8 `emulator`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

Use an emulator or surrogate model to evaluate the likelihood function

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
			gaussian_process	Gaussian Process surrogate model
	Required (<i>Choose One</i>)	Emulator Type (Group 1)		
			pce	Polynomial Chaos Expansion surrogate model
			ml_pce	Multilevel Polynomial Chaos Expansion as an emulator model.
			mf_pce	Multifidelity Polynomial Chaos Expansion as an emulator model.
			sc	Stochastic Collocation as an emulator model.
			mf_sc	Multifidelity Stochastic Collocation as an emulator model.

Description

This keyword describes the type of emulator used when calculating the likelihood function for the Bayesian calibration. The emulator can be a Gaussian process, polynomial chaos expansion, or stochastic collocation.

7.2.68.4.8.1 [gaussian_process](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Default: Surfpack Gaussian process

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	GP Implementation (Group 1)	surfpack	Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate
	Optional		build_samples	Number of initial model evaluations used in build phase
	Optional		posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.2.68.4.8.2 surfpack

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.68.4.8.3 dakota

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the `Surfpack` GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.68.4.8.4 build_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [build_samples](#)

Number of initial model evaluations used in build phase

Specification

Alias: none

Argument(s): INTEGER

Description

The number of build points or training points used in the initial phase of an algorithm or model construction. Typically these are the initial set of data points used to construct (train) a surrogate model (emulator). If the number of `build_samples` is less than the minimum number of points required to build the surrogate, Dakota will augment the samples to obtain the minimum required.

Examples

Perform GP-based adaptive importance sampling, building the GP with 100 points and then performing 100 approximate evaluations to evaluate the probability.

```
method
  gpais
    build_samples = 100
    samples_on_emulator = 100
    max_iterations = 5
    response_levels = -1.065
```

7.2.68.4.8.5 posterior_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```

bayes_calibration queso
  chain_samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack build_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives

```

7.2.68.4.8.6 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.4.8.7 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	header	Enable header row in custom-annotated tabular file
	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1         0.0002  0.26            0.76
2              0.90009    1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991    1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.4.8.8 header

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.4.8.9 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.4.8.10 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.68.4.8.11 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.12 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [gaussian_process](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.13 `active_only`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [gaussian_process](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.4.8.14 `pce`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)

Polynomial Chaos Expansion surrogate model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional	Group 1	p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Required (<i>Choose One</i>)		quadrature_order	Order for tensor-products of Gaussian quadrature rules
			sparse_grid_level	Level to use in sparse grid integration or interpolation
			cubature_integrand	Cubature using Stroud rules and their extensions
			expansion_order	The (initial) order of a polynomial expansion
			orthogonal_least_ - interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion- _file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal - covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a polynomial chaos expansion (PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)

7.2.68.4.8.15 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	<code>uniform</code>	Refine an expansion uniformly in all dimensions.
			<code>dimension_adaptive</code>	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.4.8.16 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.4.8.17 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.4.8.18 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,  
  polynomial_chaos  
    sparse_grid_level = 3  
    dimension_adaptive p_refinement sobol  
      max_iterations = 20  
      convergence_tol = 1.e-4
```

7.2.68.4.8.19 decay

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.20 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.21 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [dream](#)
- [emulator](#)
- [pce](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive generalized  
  p_refinement  
    max_refinement_iterations = 20  
    convergence_tol = 1.e-4  
    sparse_grid_level = 1
```

7.2.68.4.8.22 quadrature_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_ - preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`, where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and `method-multifidelity_stoch_collocation-quadrature_order_sequence`

7.2.68.4.8.23 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)

- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.24 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.25 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.26 sparse_grid_level

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.68.4.8.27 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.28 `restricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.29 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed

sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.30 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.31 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.32 cubature_integrand

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [cubature_integrand](#)

Cubature using Stroud rules and their extensions

Specification

Alias: none

Argument(s): INTEGER

Description

Multi-dimensional integration by Stroud cubature rules [80] and extensions [94], as specified with `cubature_integrand`. A total-order expansion is used, where the isotropic order p of the expansion is half of the integrand order, rounded down. The total number of terms N for an isotropic total-order expansion of order p over n variables is given by

$$N = 1 + P = 1 + \sum_{s=1}^p \frac{1}{s!} \prod_{r=0}^{s-1} (n+r) = \frac{(n+p)!}{n!p!}$$

Since the maximum integrand order is currently five for normal and uniform and two for all other types, at most second- and first-order expansions, respectively, will be used. As a result, cubature is primarily useful for global sensitivity analysis, where the Sobol' indices will provide main effects and, at most, two-way interactions. In addition, the random variable set must be independent and identically distributed (*iid*), so the use of `askey` or `wiener` transformations may be required to create *iid* variable sets in the transformed space (as well as to allow usage of the higher order cubature rules for normal and uniform). Note that global sensitivity analysis often assumes uniform bounded regions, rather than precise probability distributions, so the *iid* restriction would not be problematic in that case.

7.2.68.4.8.33 expansion_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)

The (initial) order of a polynomial expansion

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.
	Required (<i>Choose One</i>)	Group 1	collocation_points	Number of collocation points used to estimate expansion coefficients
			collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.
			expansion_samples	Number of simulation samples used to estimate the expected value of a set of PCE coefficients
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions

Description

When the `expansion_order` for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product

quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with `expansion_samples`). In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

Linear regression (specified with either `collocation_points` or `collocation_ratio`). A total-order expansion is used and must be specified using `expansion_order` as described in the previous option. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling ("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the `tensor_grid` keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If `collocation_points` or `collocation_ratio` is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the `noise_tolerance`, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders ($1, \dots, \text{expansion_order}$) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

7.2.68.4.8.34 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.35 basis_type

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.

			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.
--	--	--	-------------------------	---

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.4.8.36 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.4.8.37 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.4.8.38 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence-limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.39 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.40 `soft_convergence_limit`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.41 `collocation_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional <i>(Choose One)</i>	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
			basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
			basis_pursuit_denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
			least_angle_regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
			least_absolute_shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.

	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation-

_ratio-collocation_points_sequence

7.2.68.4.8.42 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group LSQ Regression Approach (Group 1)	Dakota Keyword svd	Dakota Keyword Description Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.4.8.43 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.4.8.44 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.4.8.45 `orthogonal_matching_pursuit`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.4.8.46 `noise_tolerance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.47 `basis_pursuit`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.4.8.48 `basis_pursuit_denoising`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: `bpdn`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.4.8.49 `noise_tolerance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [basis_pursuit_denoising](#)

- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.50 `least_angle_regression`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.4.8.51 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.52 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors. The l_2 penalty used when performing compressed sensing with elastic net.
	Optional		l2_penalty	

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.4.8.53 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.54 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.4.8.55 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.4.8.56 [noise_only](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.4.8.57 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.4.8.58 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.4.8.59 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.4.8.60 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, it's use with Surfpack Gaussian process is not recommended.

7.2.68.4.8.61 tensor_grid

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.4.8.62 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.63 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_points](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.4.8.64 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares
			orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
			basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
			basis_pursuit_denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

		least_angle_regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.
	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

	Optional	max_solver_ - iterations	Maximum iterations in determining polynomial coefficients
--	-----------------	--	---

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p, the collocation_ratio allows for specification of a constant factor applied to N (e.g., collocation_ratio = 2. produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using ratio_order. In this case, the number of samples becomes cN^o where c is the collocation_ratio and o is the ratio_order. The use_derivatives flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.4.8.65 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.
--	--	--	---------------------------------------	--

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.4.8.66 `svd`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.4.8.67 `equality_constrained`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)

- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.4.8.68 [orthogonal_matching_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.4.8.69 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.70 basis_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.4.8.71 `basis_pursuit_denoising`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.4.8.72 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.73 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.4.8.74 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.75 [least_absolute_shrinkage](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.4.8.76 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.77 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.4.8.78 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)

- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.4.8.79 [noise_only](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.4.8.80 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.4.8.81 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.4.8.82 `response_scaling`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.4.8.83 `use_derivatives`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.4.8.84 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.4.8.85 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.86 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.4.8.87 expansion_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [expansion_samples](#)

Number of simulation samples used to estimate the expected value of a set of PCE coefficients

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Description

In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

A corresponding sequence specification is documented at, e.g., [expansion_samples_sequence](#) and [expansion_samples_sequence](#)

See Also

These keywords may also be of interest:

- [expansion_samples_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-expansion-_samples_sequence

7.2.68.4.8.88 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [expansion_samples](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.89 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.4.8.90 custom_annotated

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3             0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.4.8.91 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.4.8.92 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.93 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.94 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.95 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.96 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)

- [expansion_order](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.4.8.97 `posterior_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [expansion_order](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```

bayes_calibration queso
  chain_samples = 2000 seed = 348
  delayed_rejection
  emulator
    gaussian_process surfpack build_samples = 30
    posterior_adaptive max_iterations = 10
    proposal_covariance derivatives

```

7.2.68.4.8.98 orthogonal_least_interpolation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: least_interpolation oli

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points	Number of collocation points used to estimate expansion coefficients
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models. File containing points you wish to use to build a surrogate
	Optional	import_build_points_file	
	Optional	posterior_adaptive	Adapt emulator model to increase accuracy in high posterior probability regions

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.4.8.99 `collocation_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points](#)

Number of collocation points used to estimate expansion coefficients

Specification

Alias: none

Argument(s): INTEGER

Description

Specify the number of collocation points used to estimate expansion coefficients using regression approaches.

A corresponding sequence specification is documented at, e.g., [collocation_points_sequence](#) and [collocation_points_sequence](#)

See Also

These keywords may also be of interest:

- [collocation_points_sequence](#) method-multilevel_polynomial_chaos-expansion_order_sequence-collocation_ratio-collocation_points_sequence

7.2.68.4.8.100 tensor_grid

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.4.8.101 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.102 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

	Optional	active_only	Import only active variables from tabular data file
--	-----------------	-----------------------------	---

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.4.8.103 custom_annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.68.4.8.104 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.4.8.105 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.106 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.107 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)

- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.108 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.109 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.4.8.110 posterior_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [orthogonal_least_interpolation](#)
- [posterior_adaptive](#)

Adapt emulator model to increase accuracy in high posterior probability regions

Specification

Alias: none

Argument(s): none

Description

Following an emulator-based MCMC process, this option refines the emulator by selecting points in regions of high posterior probability, performing truth evaluations at these points, updating the emulator, and reperforming the MCMC process. The adaptation is continued until the maximum number of iterations is exceeded or the convergence tolerance is met.

Examples

```
bayes_calibration queso
chain_samples = 2000 seed = 348
delayed_rejection
emulator
  gaussian_process surfpack build_samples = 30
  posterior_adaptive max_iterations = 10
  proposal_covariance derivatives
```

7.2.68.4.8.111 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.4.8.112 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.4.8.113 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.4.8.114 export_expansion_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.4.8.115 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.4.8.116 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.4.8.117 `ml_pce`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)

Multilevel Polynomial Chaos Expansion as an emulator model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		allocation_control	Sample allocation approach for multilevel expansions
	Optional		discrepancy_emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)	Group 1	expansion_order_sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multilevel polynomial chaos expansion (ML PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using ML PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multilevel_polynomial_chaos](#)

7.2.68.4.8.118 [allocation_control](#)

- [Keywords Area](#)

- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)

Sample allocation approach for multilevel expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Multilevel Sample Allocation Control (Group 1)	Dakota Keyword estimator_variance	Dakota Keyword Description Variance of mean estimator within multilevel polynomial chaos
			rip_sampling	Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Description

Multilevel expansions, including regression-based polynomial chaos expansion (PCE) and function train (FT) expansions, require a sample allocation strategy. Three options are currently available:

- allocation based on assuming a convergence rate for the estimator variance (for regression PCE)
- restricted isometry property (RIP) sampling (for regression PCE via compressed sensing)
- rank sampling (for FT)

Default Behavior

Current defaults, when `allocation_control` is not specified, are estimator variance for PCE and rank sampling for FT.

7.2.68.4.8.119 `estimator_variance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)

- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [estimator_variance](#)

Variance of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		estimator_rate	Rate of convergence of mean estimator within multilevel polynomial chaos

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ , with default values that may be overridden as part of this specification block.

This approach is supported for regression-based PCE approaches (over-determined least squares, under-determined compressed sensing, or orthogonal least interpolation).

In practice, it can be challenging to estimate a smooth convergence rate for estimator variance in the presence of abrupt transitions in the quality of sparse recoveries. As a result, sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the convergence of the estimator variance, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```

method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
  orthogonal_matching_pursuit
  expansion_order_sequence = 2
  pilot_samples = 10
  collocation_ratio = .9
  allocation_control
  estimator_variance estimator_rate = 2.5
  seed = 1237
  convergence_tolerance = .01

```

7.2.68.4.8.120 estimator_rate

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [estimator_variance](#)
- [estimator_rate](#)

Rate of convergence of mean estimator within multilevel polynomial chaos

Specification

Alias: none

Argument(s): REAL

Default: 2

Description

Multilevel Monte Carlo performs optimal resource allocation based on a known estimator variance for the mean statistic:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{N}$$

Replacing the simple ensemble average estimator in Monte Carlo with a polynomial chaos estimator results in a different and unknown relationship between the estimator variance and the number of samples. In one approach to multilevel PCE, we can employ a parameterized estimator variance:

$$\text{Var}[\hat{Q}] = \frac{\sigma_Q^2}{\gamma N^\kappa}$$

for free parameters γ and κ .

The default values are $\gamma = 1$ and $\kappa = 2$ (adopts a more aggressive sample profile by assuming a faster convergence rate than Monte Carlo). This advanced specification option allows to user to specify κ , overriding the default.

7.2.68.4.8.121 rip_sampling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [allocation_control](#)
- [rip_sampling](#)

Sample allocation based on restricted isometry property (RIP) within multilevel polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multilevel polynomial chaos with compressed sensing may allocate the number of samples per level based on the restricted isometry property (RIP), applied to recovery at level l :

$$N_l \geq s_l \log^3(s_l) L_l \log(C_l)$$

for sparsity s , cardinality C , and mutual coherence L . The adaptive algorithm starts from a pilot sample, shapes the profile based on observed sparsity, and iterates until convergence. In practice, RIP sampling levels are quite conservative, and a collocation ratio constraint ($N_l \leq rC_l$, where r defaults to 2) must be enforced on the profile.

The algorithm relies on observed sparsity, it is appropriate for use with regularized solvers for compressed sensing. It employs orthogonal matching pursuit (OMP) by default and automatically activates cross-validation in order to choose the best noise parameter value for the recovery.

This capability is **experimental**. Sample allocation by greedy refinement is generally preferred.

Examples

This example starts with sparse recovery for a second-order candidate expansion at each level. As the number of samples is adapted for each level, as dictated by the number of sparse coefficient sets recovered for each level, the candidate expansion order is incremented as needed in order to synchronize with the specified collocation ratio.

```
method,
  model_pointer = 'HIERARCH'
  multilevel_polynomial_chaos
  orthogonal_matching_pursuit
  expansion_order_sequence = 2
  pilot_samples = 10
  collocation_ratio = .9
  allocation_control rip_sampling
  seed = 1237
  convergence_tolerance = .01
```

7.2.68.4.8.122 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword distinct	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,  
  multilevel_polynomial_chaos  
  expansion_order_sequence = 2  
  collocation_ratio = .9  
  orthogonal_matching_pursuit  
  discrepancy_emulation recursive
```

7.2.68.4.8.123 distinct

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.4.8.124 recursive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.4.8.125 `expansion_order_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)

	Optional	Group 1	basis_type collocation_ratio	<p>Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.</p> <p>Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.</p>
			expansion_samples_sequence	<p>Sequence of expansion samples used in a multi-stage polynomial chaos expansion. Each level entry of the <code>expansion_samples_sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach. A corresponding scalar specification is documented at, e.g., expansion_samples</p>

	Optional	import_build_points_file	File containing points you wish to use to build a surrogate
--	-----------------	--	---

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.68.4.8.126 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.127 basis_type

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.4.8.128 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.4.8.129 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.4.8.130 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.131 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.132 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.133 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares

		orthogonal_-matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit_-denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle_-regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_-shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.4.8.134 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.4.8.135 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.4.8.136 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.4.8.137 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.4.8.138 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.4.8.139 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.140 [basis_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.4.8.141 [basis_pursuit_denoising](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.4.8.142 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.143 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.4.8.144 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.145 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.4.8.146 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.147 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.4.8.148 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.4.8.149 noise_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.4.8.150 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.4.8.151 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.4.8.152 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.4.8.153 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.4.8.154 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.4.8.155 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.156 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.4.8.157 `expansion_samples_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.68.4.8.158 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.159 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification**Alias:** import_points_file**Argument(s):** STRING**Default:** no point import from a file**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.4.8.160 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)

- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002      0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.161 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.4.8.162 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.4.8.163 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.164 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.165 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.166 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)

- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.4.8.167 `orthogonal_least_interpolation`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: `least_interpolation oli`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.4.8.168 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.4.8.169 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.4.8.170 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)

- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.171 `import_build_points_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional <i>(Choose One)</i>	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.4.8.172 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.4.8.173 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.4.8.174 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)

- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.175 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.176 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading eval_id and interface_id columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26         0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.177 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...

```

7.2.68.4.8.178 `active_only`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.4.8.179 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.4.8.180 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.4.8.181 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.4.8.182 `export_expansion_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.4.8.183 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.4.8.184 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [ml_pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.4.8.185 `mf_pce`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)

Multifidelity Polynomial Chaos Expansion as an emulator model.

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional	Group 1	p_refinement	Automatic polynomial order refinement
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_ - emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)		quadrature_order_ - sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_ - sequence	Sequence of sparse grid levels used in a multi-stage expansion
			expansion_order_ - sequence	Sequence of expansion orders used in a multi-stage expansion
			orthogonal_least_ - interpolation	Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 2)	askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
Dakota Version 6.16 Reference Manual generated on May 12, 2022				

	Optional		normalized	The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials
	Optional		export_expansion_file	Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file
	Optional (<i>Choose One</i>)	Covariance Type (Group 3)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multifidelity polynomial chaos expansion (MF PCE) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using MF PCE as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)

7.2.68.4.8.186 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.4.8.187 `uniform`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [mf_pce](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed.

7.2.68.4.8.188 `dimension_adaptive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			decay	Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.4.8.189 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,  
    polynomial_chaos  
    sparse_grid_level = 3  
    dimension_adaptive p_refinement sobol  
    max_iterations = 20  
    convergence_tol = 1.e-4
```

7.2.68.4.8.190 decay

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [decay](#)

Estimate spectral coefficient decay rates to guide dimension-adaptive refinement.

Specification

Alias: none

Argument(s): none

Description

Estimate spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation. These decay rates are used to guide dimension-adaptive refinement, where slower decay rates result in higher dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement decay
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.191 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.192 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,  
  polynomial_chaos  
  dimension_adaptive generalized  
  p_refinement  
    max_refinement_iterations = 20  
    convergence_tol = 1.e-4  
    sparse_grid_level = 1
```

7.2.68.4.8.193 allocation_control

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity polynomial chaos

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.68.4.8.194 [greedy](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity polynomial chaos

Specification

Alias: none

Argument(s): none

Description

Multifidelity polynomial chaos supports greedy refinement strategies, spanning regression and projection approaches for computing expansion coefficients. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within a unified competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any $z/p/\beta/\beta^*$ level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity regression starts from a zeroth-order reference expansion (a constant) for each level, and generates candidate refinements for each level that are competed in an integrated greedy competition. The number of new samples for the incremented candidate expansion order is determined from the collocation ratio. In this case, the number of candidates for each level is limited to one uniform refinement of the current expansion order.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement uniform
  expansion_order_sequence = 0
  collocation_ratio = .9 seed = 160415
  orthogonal_matching_pursuit
  convergence_tolerance 1.e-2
```

The next example employs generalized sparse grids within a greedy multifidelity competition. Each modeling level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all model levels is competed within an integrated greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each model level and the number of candidates grows rapidly with random dimension and grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_polynomial_chaos
  allocation_control greedy
  p_refinement dimension_adaptive generalized
  sparse_grid_level_sequence = 0 unrestricted
  convergence_tolerance 1.e-8
```

7.2.68.4.8.195 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Discrepancy Emulation Approach (Group 1)	distinct	Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.68.4.8.196 `distinct`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.4.8.197 `recursive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.4.8.198 quadrature_order_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.68.4.8.199 dimension_preference

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.200 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.201 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.202 sparse_grid_level_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 1)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 2)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.68.4.8.203 `dimension_preference`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.204 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.205 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.206 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.207 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.208 expansion_order_sequence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)

Sequence of expansion orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional		basis_type	Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.
	Required (<i>Choose One</i>)	Group 1	collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

			<p>expansion_ samples_sequence</p>	<p>Sequence of expansion samples used in a multi-stage polynomial chaos expansion. Each level entry of the <code>expansion_</code> <code>_samples_</code> <code>sequence</code> applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos. If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach. A corresponding scalar specification is documented at, e.g., expansion_ samples</p>
--	--	--	--	---

	Optional	import_build_points_file	File containing points you wish to use to build a surrogate
--	-----------------	--	---

Description

Each level entry of the `expansion_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion order sequences include multilevel or multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_order](#)

See Also

These keywords may also be of interest:

- [expansion_order](#)

7.2.68.4.8.209 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.210 basis_type

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)

Specify the type of basis truncation to be used for a Polynomial Chaos Expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	PCE Basis Type (Group 1)	tensor_product	Use a tensor-product index set to construct a polynomial chaos expansion.
			total_order	Use a total-order index set to construct a polynomial chaos expansion.
			adapted	Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Description

Specify the type of basis truncation that determines the terms that are included within a Polynomial Chaos Expansion. Options include tensor-product, total-order, and adapted.

7.2.68.4.8.211 `tensor_product`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [tensor_product](#)

Use a tensor-product index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use a tensor-product index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\max(i_1, \dots, i_d) \leq p$$

7.2.68.4.8.212 `total_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [total_order](#)

Use a total-order index set to construct a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Description

Use the traditional total-order index set to construct a polynomial chaos expansion. That is for a given order p keep all terms with a d -dimensional multi index $\mathbf{i} = (i_1, \dots, i_d)$ that satisfies

$$\sum_{k=1}^d i_k \leq p$$

7.2.68.4.8.213 adapted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		advancements	The maximum number of steps used to expand a basis step.
	Optional		soft_convergence_limit	The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.214 advancements

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [advancements](#)

The maximum number of steps used to expand a basis step.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.215 soft_convergence_limit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [basis_type](#)
- [adapted](#)
- [soft_convergence_limit](#)

The maximum number of times no improvement in cross validation error is allowed before the algorithm is terminated.

Specification

Alias: none

Argument(s): INTEGER

Description

Use adaptive basis selection to choose the basis terms in a polynomial chaos expansion. Basis selection uses compressed sensing to identify a initial set of non zero PCE coefficients. Then these non-zero terms are expanded a set number of times (we suggest 3) and compressed sensing is then applied to these three new index sets. Cross validation is used to choose the best candidate basis. The best basis is then restricted again to the non-zero terms and expanded until no improvement can be gained by adding additional terms.

7.2.68.4.8.216 collocation_ratio

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		collocation_points- _sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Regression Algorithm (Group 1)	least_squares	Compute the coefficients of a polynomial expansion using least squares

		orthogonal_matching_pursuit	Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)
		basis_pursuit	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.
		basis_pursuit_denoising	Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.
		least_angle_regression	Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.
		least_absolute_shrinkage	Compute the coefficients of a polynomial expansion by using the LASSO problem.
	Optional	cross_validation	Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.
	Optional	ratio_order	Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

	Optional	response_scaling	Perform bounds-scaling on response values prior to surrogate emulation
	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional	max_solver_iterations	Maximum iterations in determining polynomial coefficients

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.2.68.4.8.217 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: pilot_samples

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.4.8.218 least_squares

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)

Compute the coefficients of a polynomial expansion using least squares

Specification

Alias: none

Argument(s): none

Default: svd

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional(<i>Choose One</i>)	LSQ Regression Approach (Group 1)	svd	Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.
			equality_-constrained	Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Description

Compute the coefficients of a polynomial expansion using least squares. Specifically SVD-based least-squares will be used for solving over-determined systems. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares

7.2.68.4.8.219 svd

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [svd](#)

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using the singular value decomposition. When the number of model runs exceeds the number of terms in the PCE, the solution returned will be the least-squares solution, otherwise the solution will be the minimum norm solution computed using the pseudo-inverse.

7.2.68.4.8.220 equality_constrained

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_squares](#)
- [equality_constrained](#)

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

Specification

Alias: none

Argument(s): none

Description

Calculate the coefficients of a polynomial chaos expansion using equality constrained least squares.

7.2.68.4.8.221 orthogonal_matching_pursuit

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP)

Specification

Alias: omp

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion using orthogonal matching pursuit (OMP). Orthogonal matching pursuit (OMP) is a greedy algorithm that is useful when solving underdetermined linear systems.

7.2.68.4.8.222 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [orthogonal_matching_pursuit](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.223 [basis_pursuit](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

Specification

Alias: bp

Argument(s): none

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit ℓ_1 -minimization problem using linear programming.

7.2.68.4.8.224 [basis_pursuit_denoising](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

Specification

Alias: bpdn

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by solving the Basis Pursuit Denoising ℓ_1 -minimization problem using second order cone optimization.

7.2.68.4.8.225 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [basis_pursuit_denoising](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.226 least_angle_regression

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

Specification

Alias: lars

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Description

Compute the coefficients of a polynomial expansion by using the greedy least angle regression (LAR) method.

7.2.68.4.8.227 [noise_tolerance](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_angle_regression](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.228 least_absolute_shrinkage

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)

Compute the coefficients of a polynomial expansion by using the LASSO problem.

Specification

Alias: lasso

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_tolerance	The noise tolerance used when performing cross validation in the presence of noise or truncation errors.
	Optional		l2_penalty	The l_2 penalty used when performing compressed sensing with elastic net.

Description

Compute the coefficients of a polynomial expansion by using the LASSO problem.

7.2.68.4.8.229 noise_tolerance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [noise_tolerance](#)

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

Specification

Alias: none

Argument(s): REALLIST

Default: 1e-3 for BPDN, 0. otherwise (algorithms run until termination)

Description

The noise tolerance used when performing cross validation in the presence of noise or truncation errors.

7.2.68.4.8.230 l_2 _penalty

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [least_absolute_shrinkage](#)
- [l2_penalty](#)

The l_2 penalty used when performing compressed sensing with elastic net.

Specification

Alias: none

Argument(s): REAL

Default: 0. (reverts to standard LASSO formulation)

Description

The l_2 penalty used when performing compressed sensing with elastic net.

7.2.68.4.8.231 cross_validation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)

Use cross validation to choose the 'best' polynomial order of a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		noise_only	Restrict the cross validation process to estimating only the best noise tolerance.
	Optional		max_cv_order_candidates	Limit the number of cross-validation candidates for basis order

Description

Use cross validation to choose the 'best' polynomial degree of a polynomial chaos expansion. 10 fold cross validation is used to estimate the cross validation error of a total-order polynomial expansion for orders 1 through to order. The order chosen is the one that produces the lowest cross validation error. If there are not enough points to perform 10 fold cross validation then one-at-a-time cross validation will be performed.

7.2.68.4.8.232 noise_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [noise_only](#)

Restrict the cross validation process to estimating only the best noise tolerance.

Specification

Alias: none

Argument(s): none

Default: false

Description

By default, cross validation estimates both the best noise tolerance and the best candidate basis order. For reasons of reducing computational cost by reducing the number of candidate solves, the user may wish to restrict this process to estimating only the best noise tolerance.

Generally speaking, computing the best noise tolerance through cross validation mitigates issues with over-fitting the data. Computing the best candidate basis order can also mitigate overfitting, while also controlling levels of mutual coherence resulting from high-order Vandermonde-like matrix systems.

7.2.68.4.8.233 `max_cv_order_candidates`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [cross_validation](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.2.68.4.8.234 ratio_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [ratio_order](#)

Specify a non-linear the relationship between the expansion order of a polynomial chaos expansion and the number of samples that will be used to compute the PCE coefficients.

Specification

Alias: none

Argument(s): REAL

Default: 1.

Description

When using regression type methods (specified with either `collocation_points` or `collocation_ratio`), a total-order expansion can be specified using `expansion_order`. To avoid requiring the user to calculate N from n and p), the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2`. produces samples = $2N$). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`.

7.2.68.4.8.235 response_scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to $[0,1]$ based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.2.68.4.8.236 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.68.4.8.237 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.4.8.238 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.239 max_solver_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [collocation_ratio](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.2.68.4.8.240 `expansion_samples_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)

Sequence of expansion samples used in a multi-stage polynomial chaos expansion

Each level entry of the `expansion_samples_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support expansion samples sequences include multilevel and multifidelity polynomial chaos.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [expansion_samples](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

See Also

These keywords may also be of interest:

- [expansion_samples](#)

7.2.68.4.8.241 reuse_points

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [expansion_samples_sequence](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.242 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification**Alias:** import_points_file**Argument(s):** STRING**Default:** no point import from a file**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.4.8.243 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)

- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9          1.1          0.0002         0.26           0.76
2             0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3             0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.4.8.244 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.4.8.245 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `eval_id` column

Description

See description of parent `custom_annotated`

7.2.68.4.8.246 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.247 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.248 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.249 active_only

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)

- [expansion_order_sequence](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.4.8.250 `orthogonal_least_interpolation`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation.

Specification

Alias: `least_interpolation oli`

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_points_sequence	Sequence of collocation point counts used in a multi-stage expansion
	Optional		tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional		reuse_points	This describes the behavior of reuse of points in constructing polynomial chaos expansion models.
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate

Description

Build a polynomial chaos expansion from simulation samples using orthogonal least interpolation. Unlike the other regression methods `expansion_order` cannot be set. OLI will produce the lowest degree polynomial that interpolates the data

7.2.68.4.8.251 `collocation_points_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [collocation_points_sequence](#)

Sequence of collocation point counts used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Each level entry of the `collocation_points_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support collocation point sequences include multilevel and multifidelity polynomial chaos and multilevel and multifidelity function train expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [collocation_points](#)

See Also

These keywords may also be of interest:

- [collocation_points](#)

7.2.68.4.8.252 `tensor_grid`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): INTEGERLIST

Default: regression with LHS sample set (point collocation)

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.2.68.4.8.253 `reuse_points`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)

- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [reuse_points](#)

This describes the behavior of reuse of points in constructing polynomial chaos expansion models.

Specification

Alias: reuse_samples

Argument(s): none

Default: no sample reuse in coefficient estimation

Description

The `reuse_points` option controls the reuse behavior of points for various types of polynomial chaos expansions, including: `collocation_points`, `collocation_ratio`, `expansion_samples`, or `orthogonal_least_interpolation`. If any of these approaches are specified to create a set of points for the polynomial chaos expansion, one can specify `reuse_points` so that any points that have been previously generated (for example, from the `import_points` file) can be reused.

7.2.68.4.8.254 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional <i>(Choose One)</i>	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.68.4.8.255 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009       1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991       1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.4.8.256 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.68.4.8.257 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)

- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.258 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no interface_id column

Description

See description of parent `custom_annotated`

7.2.68.4.8.259 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002      0.26         0.76
2          NO_ID            0.90009    1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991    1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.8.260 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

          0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...

```

7.2.68.4.8.261 `active_only`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [orthogonal_least_interpolation](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.68.4.8.262 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.4.8.263 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.4.8.264 normalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [normalized](#)

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

Specification

Alias: none

Argument(s): none

Default: PCE coefficients correspond to unnormalized basis polynomials

Description

The normalized specification requests output of PCE coefficients that correspond to normalized orthogonal basis polynomials

7.2.68.4.8.265 `export_expansion_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [export_expansion_file](#)

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file

Specification

Alias: none

Argument(s): STRING

Description

Export the coefficients and multi-index of a Polynomial Chaos Expansion (PCE) to a file. The multi-index written will be sparse. Specifically the expansion will consist only of the indices corresponding to the non-zero coefficients of the PCE.

7.2.68.4.8.266 `diagonal_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.4.8.267 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_pce](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.4.8.268 `sc`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)

Stochastic Collocation as an emulator model.

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Required (<i>Choose One</i>)	Interpolation Grid Type (Group 2)	quadrature_order	Order for tensor-products of Gaussian quadrature rules
			sparse_grid_level	Level to use in sparse grid integration or interpolation
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.
			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects stochastic collocation (SC) model to use in the Bayesian likelihood calculations. Most specification options are carried over for using SC as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [stoch_collocation](#)

7.2.68.4.8.269 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.

			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
--	--	--	------------------------------------	--

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.4.8.270 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.4.8.271 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.
--	--	--	-----------------------------	---

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.4.8.272 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.273 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.274 h_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [dream](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group h-refinement Type (Group 1)	Dakota Keyword uniform	Dakota Keyword Description Refine an expansion uniformly in all dimensions.
			dimension_-adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either `p_refinement` or `h_refinement`, and either refinement specification can be either `uniform` or `dimension_adaptive`. The `dimension_adaptive` case can be further specified as either `sobol` or `generalized` (decay not supported). Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls. The `h_refinement` specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify `piecewise` in the case of `h_refinement`). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current `uniform` and `dimension_adaptive` options.

7.2.68.4.8.275 `uniform`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.4.8.276 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required (<i>Choose One</i>)	Dimension Adaptivity Estimation Approach (Group 1)	sobol	Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.4.8.277 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.278 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```


7.2.68.4.8.279 local_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.68.4.8.280 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.68.4.8.281 quadrature_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)

Order for tensor-products of Gaussian quadrature rules

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multidimensional integration by a tensor-product of Gaussian quadrature rules (specified with `quadrature_order`, and, optionally, `dimension_preference`). The default rule selection is to employ `non_nested` Gauss rules including Gauss-Hermite (for normals or transformed normals), Gauss-Legendre (for uniforms or transformed uniforms), Gauss-Jacobi (for betas), Gauss-Laguerre (for exponentials), generalized Gauss-Laguerre (for gammas), and numerically-generated Gauss rules (for other distributions when using an Extended basis). For the case of `p_refinement` or the case of an explicit `nested` override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse. By specifying a `dimension_preference`,

where higher preference leads to higher order polynomial resolution, the tensor grid may be rendered anisotropic. The dimension specified to have highest preference will be set to the specified `quadrature_order` and all other dimensions will be reduced in proportion to their reduced preference; any non-integral portion is truncated. To synchronize with tensor-product integration, a tensor-product expansion is used, where the order p_i of the expansion in each dimension is selected to be half of the integrand precision available from the rule in use, rounded down. In the case of non-nested Gauss rules with integrand precision $2m_i - 1$, p_i is one less than the quadrature order m_i in each dimension (a one-dimensional expansion contains the same number of terms, $p + 1$, as the number of Gauss points). The total number of terms, N , in a tensor-product expansion involving n uncertain input variables is

$$N = 1 + P = \prod_{i=1}^n (p_i + 1)$$

In some advanced use cases (e.g., multifidelity UQ), multiple grid resolutions can be employed; for this reason, the `quadrature_order` specification supports an array input.

A corresponding sequence specification is documented at, e.g., [quadrature_order_sequence](#) and [quadrature_order_sequence](#)

See Also

These keywords may also be of interest:

- [quadrature_order_sequence](#) and [method-multifidelity_stoch_collocation-quadrature_order_sequence](#)

7.2.68.4.8.282 dimension_preference

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.283 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.284 `non_nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [quadrature_order](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.285 `sparse_grid_level`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation

			hierarchical	Employ a hierarchical sparse grid construction
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with `sparse_grid_level` and, optionally, `dimension_preference`). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit `non_nested` specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the `dimension_preference` specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full `sparse_grid_level` and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for `quadrature_order`, the `sparse_grid_level` specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.2.68.4.8.286 `dimension_preference`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.287 nodal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.68.4.8.288 hierarchical

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.68.4.8.289 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.290 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed

sequences." To override this default behavior in the case of sparse grids, the unrestricted keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the dimension_adaptive p_refinement generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.291 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.292 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [sparse_grid_level](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.293 piecewise

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.68.4.8.294 askey

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.4.8.295 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.4.8.296 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, it's use with Surpack Gaussian process is not recommended.

7.2.68.4.8.297 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.4.8.298 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [sc](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.4.8.299 `mf_sc`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)

Multifidelity Stochastic Collocation as an emulator model.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Automated Refinement Type (Group 1)	p_refinement	Automatic polynomial order refinement
			h_refinement	Employ h-refinement to refine the grid
	Optional		max_refinement_ - iterations	Maximum number of expansion refinement iterations
	Optional		allocation_control	Sample allocation approach for multifidelity expansions
	Optional		discrepancy_ - emulation	Formulation for emulation of model discrepancies.
	Required (<i>Choose One</i>)	Interpolation Grid Type (Group 2)	quadrature_order_ - sequence	Sequence of quadrature orders used in a multi-stage expansion
			sparse_grid_level_ - sequence	Sequence of sparse grid levels used in a multi-stage expansion
	Optional (<i>Choose One</i>)	Basis Polynomial Family (Group 3)	piecewise	Use piecewise local basis functions
			askey	Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

			wiener	Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional (<i>Choose One</i>)	Covariance Type (Group 4)	diagonal_covariance	Display only the diagonal terms of the covariance matrix
			full_covariance	Display the full covariance matrix

Description

Selects a multifidelity stochastic collocation (MF SC) surrogate model to use in the Bayesian likelihood calculations. Most specification options are carried over for using MF SC as a surrogate within the Bayesian framework.

See Also

These keywords may also be of interest:

- [multifidelity_stoch_collocation](#)

7.2.68.4.8.300 p_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)

Automatic polynomial order refinement

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	p-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Description

The `p_refinement` keyword specifies the usage of automated polynomial order refinement, which can be either `uniform` or `dimension_adaptive`.

The `dimension_adaptive` option is supported for the tensor-product quadrature and Smolyak sparse grid options and `uniform` is supported for tensor and sparse grids as well as regression approaches (`collocation_points` or `collocation_ratio`).

Each of these refinement cases makes use of the `max_iterations` and `convergence_tolerance` method independent controls. The former control limits the number of refinement iterations, and the latter control terminates refinement when the two-norm of the change in the response covariance matrix (or, in goal-oriented approaches, the two-norm of change in the statistical quantities of interest (QOI)) falls below the tolerance.

The `dimension_adaptive` case can be further specified to utilize `sobol`, `decay`, or `generalized` refinement controls. The former two cases employ anisotropic tensor/sparse grids in which the anisotropic dimension preference (leading to anisotropic integrations/expansions with differing refinement levels for different random dimensions) is determined using either total Sobol' indices from variance-based decomposition (`sobol` case: high indices result in high dimension preference) or using spectral coefficient decay rates from a rate estimation technique similar to Richardson extrapolation (`decay` case: low decay rates result in high dimension preference). In these two cases as well as the `uniform` refinement case, the `quadrature_order` or `sparse_grid_level` are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the `uniform` refinement case with regression approaches, the `expansion_order` is ramped by one on each iteration while the oversampling ratio (either defined by `collocation_ratio` or inferred from `collocation_points` based on the initial expansion) is held fixed. Finally, the `generalized dimension_adaptive` case is the default adaptive approach; it refers to the generalized sparse grid algorithm, a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

For the case of `p_refinement` or the case of an explicit nested override, Gauss-Hermite rules are replaced with Genz-Keister nested rules and Gauss-Legendre rules are replaced with Gauss-Patterson nested rules, both of which exchange lower integrand precision for greater point reuse.

7.2.68.4.8.301 `uniform`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)

- [mf_sc](#)
- [p_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.4.8.302 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either sobol or generalized for stochastic collocation and either sobol, decay, or generalized for polynomial chaos. Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls.

7.2.68.4.8.303 sobol

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: generalized

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.304 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [p_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.305 h_refinement

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)

Employ h-refinement to refine the grid

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	h-refinement Type (Group 1)	uniform	Refine an expansion uniformly in all dimensions.
			dimension_adaptive	Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.
			local_adaptive	Planned future capability for local pointwise refinement within a generalized sparse grid.

Description

Automated expansion refinement can be selected as either p_refinement or h_refinement, and either refinement specification can be either uniform or dimension_adaptive. The dimension_adaptive case can be further specified as either sobol or generalized (decay not supported). Each of these automated refinement approaches makes use of the max_iterations and convergence_tolerance iteration controls. The h_refinement specification involves use of the same piecewise interpolants (linear or cubic Hermite splines) described above for the piecewise specification option (it is not necessary to redundantly specify piecewise in the case of h_refinement). In future releases, the hierarchical interpolation approach will enable local refinement in addition to the current uniform and dimension_adaptive options.

7.2.68.4.8.306 uniform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [uniform](#)

Refine an expansion uniformly in all dimensions.

Specification

Alias: none

Argument(s): none

Description

The quadrature_order or sparse_grid_level are ramped by one on each refinement iteration until either of the two convergence controls is satisfied. For the uniform refinement case with regression approaches, the expansion_order is ramped by one on each iteration while the oversampling ratio (either defined by collocation_ratio or inferred from collocation_points based on the initial expansion) is held fixed.

7.2.68.4.8.307 dimension_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to have higher 'importance'.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Dimension Adaptivity Estimation Approach (Group 1)	Dakota Keyword sobol	Dakota Keyword Description Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.
			generalized	Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Description

Perform anisotropic expansion refinement by preferentially adapting in dimensions that are detected to hold higher 'importance' in resolving statistical quantities of interest.

Dimension importance must be estimated as part of the refinement process. Techniques include either [sobol](#) or [generalized](#) for stochastic collocation and either [sobol](#), [decay](#), or [generalized](#) for polynomial chaos. Each of these automated refinement approaches makes use of the `max_iterations` and `convergence_tolerance` iteration controls.

7.2.68.4.8.308 [sobol](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [sobol](#)

Estimate dimension preference for automated refinement of stochastic expansion using total Sobol' sensitivity indices.

Specification

Alias: none

Argument(s): none

Default: [generalized](#)

Description

Determine dimension preference for refinement of a stochastic expansion from the total Sobol' sensitivity indices obtained from global sensitivity analysis. High indices indicate high importance for resolving statistical quantities of interest and therefore result in high dimension preference.

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement sobol
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.309 generalized

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [dimension_adaptive](#)
- [generalized](#)

Use the generalized sparse grid dimension adaptive algorithm to refine a sparse grid approximation of stochastic expansion.

Specification

Alias: none

Argument(s): none

Description

The generalized sparse grid algorithm is a greedy approach in which candidate index sets are evaluated for their impact on the statistical QOI, the most influential sets are selected and used to generate additional candidates, and the index set frontier of a sparse grid is evolved in an unstructured and goal-oriented manner (refer to User's Manual PCE descriptions for additional specifics).

Examples

```
method,
  polynomial_chaos
  sparse_grid_level = 3
  dimension_adaptive p_refinement generalized
  max_iterations = 20
  convergence_tol = 1.e-4
```

7.2.68.4.8.310 local_adaptive

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [h_refinement](#)
- [local_adaptive](#)

Planned future capability for local pointwise refinement within a generalized sparse grid.

Specification

Alias: none

Argument(s): none

Description

Sparse grid interpolation admits approaches for pointwise local refinement within the general framework of generalized sparse grids. This algorithmic capability is currently in a partial prototype stage.

7.2.68.4.8.311 max_refinement_iterations

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [max_refinement_iterations](#)

Maximum number of expansion refinement iterations

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

Limits the number of times the expansion can be refined under various refinement strategies.

Examples

```
method,
  polynomial_chaos
  dimension_adaptive generalized
  p_refinement
  max_refinement_iterations = 20
  convergence_tol = 1.e-4
  sparse_grid_level = 1
```

7.2.68.4.8.312 allocation_control

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [allocation_control](#)

Sample allocation approach for multifidelity expansions

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Multifidelity Sample Allocation Control (Group 1)	greedy	Sample allocation based on greedy refinement within multifidelity stochastic collocation

Description

Multifidelity surrogate approaches, including polynomial chaos, stochastic collocation, and function train, can optionally employ an integrated greedy competition across the model sequence, where each model index can supply one or more refinement candidates which are competed to determine the candidate with the greatest impact on the QoI statistics per unit cost. This greedy competition implicitly determines the optimal sample allocation across model indices.

Default Behavior

The default, when `allocation_control` is not specified, is to compute or adapt separately for each model index (individual instead of integrated refinement).

7.2.68.4.8.313 greedy

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [allocation_control](#)
- [greedy](#)

Sample allocation based on greedy refinement within multifidelity stochastic collocation

Specification

Alias: none

Argument(s): none

Description

Multifidelity stochastic collocation supports greedy refinement strategies using tensor and sparse grids for both nodal and hierarchical collocation approaches. The key idea is that each level of the model hierarchy being approximated can generate one or more candidates for refinement. These candidates are competed against each other within an integrated competition, and the candidate that induces the largest change in the statistical QoI (response covariance by default, or results of any level mappings when specified), normalized by relative cost of evaluating the candidate, is selected and then used to generate additional candidates for consideration at its model level.

Examples

The following example of greedy multifidelity stochastic collocation using nodal interpolation starts from a zeroth-order expansion (a constant) for each level, and generates uniform candidate refinements for each level that are competed in a greedy competition. The number of new samples for the incremented candidate expansion order is determined from the quadrature rules of the new sparse grid level. In this case, the number of candidates for each level is limited to one uniform refinement of the current sparse grid level.

```
method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
    nodal
    allocation_control greedy
    p_refinement uniform
    sparse_grid_level_sequence = 0 unrestricted
    convergence_tolerance 1.e-3
```

The next example employs generalized sparse grids and hierarchical interpolation. Each level starts from a level 0 reference grid (a single point) and generates multiple admissible index set candidates. The full set of candidates across all levels is competed within a unified greedy competition, where the greedy selection metric is the induced change in the statistical QoI, normalized by the aggregate simulation cost of the index set candidate. In this case, there are multiple candidates for each level and the number of candidates grows rapidly with random dimension and grid level.

```

method,
  model_pointer = 'HIERARCH'
  multifidelity_stoch_collocation
  hierarchical
  allocation_control greedy
  p_refinement dimension_adaptive generalized
  sparse_grid_level_sequence = 0 unrestricted
  convergence_tolerance 1.e-8

```

7.2.68.4.8.314 discrepancy_emulation

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)

Formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Default: distinct

Child Keywords:

	Required/Optional <i>Required(Choose One)</i>	Description of Group Discrepancy Emulation Approach (Group 1)	Dakota Keyword distinct	Dakota Keyword Description Distinct formulation for emulation of model discrepancies.
			recursive	Recursive formulation for emulation of model discrepancies.

Description

In many uncertainty quantification approaches, model discrepancies are emulated using, e.g., polynomial chaos, stochastic collocation, or Gaussian process models. Two formulations are available for this emulation:

1. `distinct` emulation (default), in which we directly approximate the difference or ratio between the evaluations of two models or solution levels.
2. `recursive` emulation (experimental option), in which we approximate a difference or ratio among the new model evaluation and the emulator approximation of the previous model.

The latter approach is a form of hierarchical emulation in which we emulate the surplus between the previous emulator and the new modeling level. This approach has a few advantages: (i) it reduces bias by correcting for emulation errors that occur at previous levels, and (ii) it does not require paired model evaluations for each discrepancy level, which reduces cost, allows for disparate sample points, and simplifies data imports.

On the other hand, its primary disadvantage is that the aggregate emulation is only as good as its weakest link, in that a poor emulator recovery can create difficulty in accurately resolving discrepancies that are recursively dependent on it. Thus, the `distinct` approach may tend to be more expensive in exchange for greater robustness.

Examples

```
method,
  multilevel_polynomial_chaos
  expansion_order_sequence = 2
  collocation_ratio = .9
  orthogonal_matching_pursuit
  discrepancy_emulation recursive
```

7.2.68.4.8.315 `distinct`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)
- [distinct](#)

Distinct formulation for emulation of model discrepancies.

Specification

Alias: paired

Argument(s): none

Description

The `distinct` approach to discrepancy emulation directly approximate the difference or ratio between the evaluations of two models or solution levels. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.4.8.316 `recursive`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [discrepancy_emulation](#)
- [recursive](#)

Recursive formulation for emulation of model discrepancies.

Specification

Alias: none

Argument(s): none

Description

The `recursive` approach to discrepancy emulation approximates a difference or ratio among the new model evaluation and the emulator approximation of the previous model. Refer to the parent documentation node for additional discussion.

Examples

7.2.68.4.8.317 `quadrature_order_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)

Sequence of quadrature orders used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		dimension_ - preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 1)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `quadrature_order_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support quadrature order sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [quadrature_order](#)

See Also

These keywords may also be of interest:

- [quadrature_order](#)

7.2.68.4.8.318 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.319 `nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: `non_nested` unless automated refinement; sparse grids: `nested`

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.320 `non_nested`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [quadrature_order_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.321 `sparse_grid_level_sequence`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)

Sequence of sparse grid levels used in a multi-stage expansion

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)
	Optional (<i>Choose One</i>)	Group 1	nodal	Employ a nodal sparse grid construction in stochastic collocation

			hierarchical	Employ a hierarchical sparse grid construction
	Optional (<i>Choose One</i>)	Quadrature Rule Growth (Group 2)	restricted	Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.
			unrestricted	Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.
	Optional (<i>Choose One</i>)	Quadrature Rule Nesting (Group 3)	nested	Enforce use of nested quadrature rules if available
			non_nested	Enforce use of non-nested quadrature rules

Description

Each level entry of the `sparse_grid_level_sequence` applies to one expansion within a multi-stage expansion. Current multi-stage expansions that support sparse grid level sequences include multilevel polynomial chaos, multifidelity polynomial chaos, or multifidelity stochastic collocation expansions.

If adaptive refinement is active, then this sequence specifies the starting point for each level within either an individual or integrated refinement approach.

A corresponding scalar specification is documented at, e.g., [sparse_grid_level](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level](#)

7.2.68.4.8.322 `dimension_preference`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.2.68.4.8.323 nodal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [nodal](#)

Employ a nodal sparse grid construction in stochastic collocation

Specification

Alias: none

Argument(s): none

Default: nodal

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The nodal formulation does not require nested interpolation rules and is therefore more generally applicable. However, in the context of adaptive refinement, it can suffer from loss of precision due to subtractive cancellation.

7.2.68.4.8.324 hierarchical

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [hierarchical](#)

Employ a hierarchical sparse grid construction

Specification

Alias: none

Argument(s): none

Description

Stochastic collocation with sparse grids can employ either a nodal or hierarchical formulation. The hierarchical formulation requires nested interpolation rules as it utilizes increments in collocation points along with hierarchical weights defined from a hierarchical basis formulation.

This keyword requests the hierarchical formulation, which has benefits for adaptive refinement by supporting direct computation of increments in targeted statistics (e.g. mean, covariance, reliability indices), avoiding loss of precision from subtractive cancellation.

7.2.68.4.8.325 restricted

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [restricted](#)

Restrict the growth rates for nested and non-nested rules can be synchronized for consistency.

Specification

Alias: none

Argument(s): none

Default: restricted (except for generalized sparse grids)

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.326 `unrestricted`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [unrestricted](#)

Override the default restriction of growth rates for nested and non-nested rules that are by default synchronized for consistency.

Specification

Alias: none

Argument(s): none

Description

In the quadrature and sparse grid cases, growth rates for nested and non-nested rules can be synchronized for consistency. For a non-nested Gauss rule used within a sparse grid, linear one-dimensional growth rules of $m = 2l + 1$ are used to enforce odd quadrature orders, where l is the grid level and m is the number of points in the rule. The precision of this Gauss rule is then $i = 2m - 1 = 4l + 1$. For nested rules, order growth with level is typically exponential; however, the default behavior is to restrict the number of points to be the lowest order rule that is available that meets the one-dimensional precision requirement implied by either a level l for a sparse grid ($i = 4l + 1$) or an order m for a tensor grid ($i = 2m - 1$). This behavior is known as "restricted growth" or "delayed sequences." To override this default behavior in the case of sparse grids, the `unrestricted` keyword can be used; it cannot be overridden for tensor grids using nested rules since it also provides a mapping to the available nested rule quadrature orders. An exception to the default usage of restricted growth is the `dimension_adaptive p_refinement` generalized sparse grid case described previously, since the ability to evolve the index sets of a sparse grid in an unstructured manner eliminates the motivation for restricting the exponential growth of nested rules.

7.2.68.4.8.327 nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [nested](#)

Enforce use of nested quadrature rules if available

Specification

Alias: none

Argument(s): none

Default: quadrature: non_nested unless automated refinement; sparse grids: nested

Description

Enforce use of nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the Nested Genz-Keister rule instead of the default non-nested Gauss-Hermite rule variables are

7.2.68.4.8.328 non_nested

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [sparse_grid_level_sequence](#)
- [non_nested](#)

Enforce use of non-nested quadrature rules

Specification

Alias: none

Argument(s): none

Description

Enforce use of non-nested quadrature rules if available. For instance if the aleatory variables are Gaussian use the non-nested Gauss-Hermite rule

7.2.68.4.8.329 `piecewise`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [piecewise](#)

Use piecewise local basis functions

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

SC also supports the option of piecewise local basis functions. These are piecewise linear splines, or in the case of gradient-enhanced interpolation via the `use_derivatives` specification, piecewise cubic Hermite splines. Both of these basis selections provide local support only over the range from the interpolated point to its nearest 1D neighbors (within a tensor grid or within each of the tensor grids underlying a sparse grid), which exchanges the fast convergence of global bases for smooth functions for robustness in the representation of nonsmooth response functions (that can induce Gibbs oscillations when using high-order global basis functions). When local basis functions are used, the usage of nonequidistant collocation points (e.g., the Gauss point selections described above) is not well motivated, so equidistant Newton-Cotes points are employed in this case, and all random variable types are transformed to standard uniform probability space.

7.2.68.4.8.330 `askey`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [askey](#)

Select the standardized random variables (and associated basis polynomials) from the Askey family that best match the user-specified random variables.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Askey option employs standard normal, standard uniform, standard exponential, standard beta, and standard gamma random variables in a transformed probability space. These selections correspond to Hermite, Legendre, Laguerre, Jacobi, and generalized Laguerre orthogonal polynomials, respectively.

Specific mappings for the basis polynomials are based on a closest match criterion, and include Hermite for normal (optimal) as well as bounded normal, lognormal, bounded lognormal, gumbel, frechet, and weibull (sub-optimal); Legendre for uniform (optimal) as well as loguniform, triangular, and bin-based histogram (sub-optimal); Laguerre for exponential (optimal); Jacobi for beta (optimal); and generalized Laguerre for gamma (optimal).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [wiener](#)

7.2.68.4.8.331 wiener

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [wiener](#)

Use standard normal random variables (along with Hermite orthogonal basis polynomials) when transforming to a standardized probability space.

Specification

Alias: none

Argument(s): none

Default: extended (Askey + numerically-generated)

Description

The Wiener option employs standard normal random variables in a transformed probability space, corresponding to a Hermite orthogonal polynomial basis. This is the same nonlinear variable transformation used by local and global reliability methods (and therefore has the same variable support).

See Also

These keywords may also be of interest:

- [polynomial_chaos](#)
- [askey](#)

7.2.68.4.8.332 use_derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, its use with Surpack Gaussian process is not recommended.

7.2.68.4.8.333 diagonal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [diagonal_covariance](#)

Display only the diagonal terms of the covariance matrix

Specification

Alias: none

Argument(s): none

Default: `diagonal_covariance` for response vector > 10 ; else `full_covariance`

Description

With a large number of responses, the covariance matrix can be very large. `diagonal_covariance` is used to suppress the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

7.2.68.4.8.334 `full_covariance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [emulator](#)
- [mf_sc](#)
- [full_covariance](#)

Display the full covariance matrix

Specification

Alias: none

Argument(s): none

Description

With a large number of responses, the covariance matrix can be very large. `full_covariance` is used to force Dakota to output the full covariance matrix.

7.2.68.4.9 `standardized_space`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [standardized_space](#)

Perform Bayesian inference in standardized probability space

Specification

Alias: none

Argument(s): none

Description

This option transforms the inference process (MCMC sampling and any emulator model management) into a standardized probability space.

The variable transformations performed are as described in [askey](#).

Default Behavior

The default for the Gaussian process and no emulator options is to perform inference in the original probability space (no transformation). Polynomial chaos and stochastic collocation emulators, on the other hand, are always formed in standardized probability space, such that the inference process is also performed in this standardized space.

Expected Output

The user will see the truth model evaluations performed in the original space, whereas any method diagnostics relating to the MCMC samples (e.g., QUESO data in the outputData directory) will report points and response data (response gradients and Hessians, if present, will differ but response values will not) that correspond to the transformed space.

Usage Tips

Selecting `standardized_space` generally has the effect of scaling the random variables to be of comparable magnitude, which can improve the efficiency of the Bayesian inference process.

Examples

```
method,
    bayes_calibration queso
    samples = 2000 seed = 348
    dram
    standardized_space
```

7.2.68.4.10 export_chain_points_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [export_chain_points_file](#)

Export the MCMC chain to the specified filename

Specification

Alias: none

Argument(s): STRING

Default: chain export to default filename

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

The filename to which the final MCMC posterior chain will be exported.

Default Behavior No export to file.

Expected Output

A tabular data file will be produced in the specified format (annotated by default) containing samples from the posterior distribution.

Usage Tips

Additional Discussion

7.2.68.4.10.1 [custom_annotated](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [export_chain_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	header	Enable header row in custom-annotated tabular file
	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1         0.0002  0.26           0.76
2              0.90009    1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991    1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.4.10.2 header

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [dream](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent [custom_annotated](#)

7.2.68.4.10.3 [eval_id](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [export_chain_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent [custom_annotated](#)

7.2.68.4.10.4 [interface_id](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [export_chain_points_file](#)

- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.4.10.5 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [export_chain_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.4.10.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [dream](#)
- [export_chain_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.2.68.5 muq

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)

Markov Chain Monte Carlo algorithms from the MUQ package

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		chain_samples	Number of Markov Chain Monte Carlo posterior samples
	Optional		seed	Seed of the random number generator
	Optional		rng	Selection of a random number generator
	Optional (<i>Choose One</i>)	MCMC Algorithm (Group 1)	dram	Use the DRAM MCMC algorithm
			delayed_rejection	Use the Delayed Rejection MCMC algorithm
			adaptive_metropolis	Use the Adaptive Metropolis MCMC algorithm
			metropolis_hastings	Use the Metropolis-Hastings MCMC algorithm
	Optional		proposal_covariance	Defines the technique used to generate the MCMC proposal covariance.

Description

The `muq` method supports the following MCMC algorithms: adaptive metropolis (AM), Metropolis Hasting (MH), delayed rejection (DR), or delayed-rejection adaptive metropolis (DRAM).

The `muq` method is currently an experimental method that relies on algorithms from MIT's MUQ code documented at: <https://bitbucket.org/mituq/muq2/src/master/>

We anticipate using more advanced features of MUQ such as Hamiltonian Monte Carlo and Langevin methods in future releases of Dakota.

7.2.68.5.1 chain_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [muq](#)
- [chain_samples](#)

Number of Markov Chain Monte Carlo posterior samples

Specification

Alias: samples

Argument(s): INTEGER

Default: method-dependent

Description

The `chain_samples` keyword indicates the number of draws from the posterior distribution to perform. When an emulator is active, this will be the number of samples on the constructed surrogate model.

Default Behavior

The default number of chain samples is method-dependent. QUESO and MUQ methods use 1000. DREAM uses (number of generations) x (number of chains), resulting in `chain_samples` close to that specified.

Usage Tips

MCMC methods typically require a large number of chain samples to converge, often thousands to millions.

Examples

```
method
  bayes_calibration queso
  chain_samples = 20000
```

7.2.68.5.2 seed

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.68.5.3 rng

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	RNG Algorithm (Group 1)	mt19937	Generates random numbers using the Mersenne twister

			<code>rnum2</code>	Generates pseudo-random numbers using the Pecos package
--	--	--	--------------------	---

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.68.5.3.1 mt19937

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.68.5.3.2 rnum2

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.68.5.4 dram

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [dram](#)

Use the DRAM MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

The type of Markov Chain Monte Carlo used. This keyword specifies the use of DRAM, (Delayed Rejection Adaptive Metropolis) [40].

Default Behavior

Five MCMC algorithm variants are supported: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Usage Tips

If the user knows very little about the proposal covariance, using `dram` is a recommended strategy. The proposal covariance is adaptively updated, and the delayed rejection may help improve low acceptance rates.

Examples

```
method,
  bayes_calibration queso
  dram
  samples = 10000 seed = 348
```

7.2.68.5.5 `delayed_rejection`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [delayed_rejection](#)

Use the Delayed Rejection MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of the Delayed Rejection algorithm in which there can be a delay in rejecting samples from the chain. That is, the "DR" part of DRAM is used but the "AM" part is not, rather a regular Metropolis-Hastings algorithm is used.

Default Behavior

Five MCMC algorithm variants are supported: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Usage Tips

If the user knows something about the proposal covariance or the proposal covariance is informed through derivative information, using `delayed_rejection` is preferred over `dram`: the proposal covariance is already being informed by derivative information and the adaptive metropolis is not necessary.

Examples

```
method,
  bayes_calibration queso
  delayed_rejection
  samples = 10000 seed = 348
```

See Also

These keywords may also be of interest:

- [proposal_covariance](#)

7.2.68.5.6 adaptive_metropolis

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [adaptive_metropolis](#)

Use the Adaptive Metropolis MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of the Adaptive Metropolis algorithm. That is, the "AM" part of DRAM is used but the "DR" part is not: specifying this keyword activates only the Adaptive Metropolis part of the MCMC algorithm, in which the covariance of the proposal density is updated adaptively.

Default Behavior

Five MCMC algorithm variants are supported in QUESO: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Four MCMC variants are currently supported in MUQ: `dram`, `delayed_rejection`, `metropolis_hastings` and `adaptive_metropolis`.

Usage Tips

If the user knows very little about the proposal covariance, but doesn't want to incur the cost of using full `dram` with both delayed rejection and adaptive metropolis, specifying only `adaptive_metropolis` offers a good strategy.

Examples

```
method,
  bayes_calibration queso
  adaptive_metropolis
  samples = 10000 seed = 348
```

7.2.68.5.7 metropolis_hastings

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [metropolis_hastings](#)

Use the Metropolis-Hastings MCMC algorithm

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Default: dram

Description

This keyword specifies the use of a Metropolis-Hastings algorithm for the MCMC chain generation. This means there is no delayed rejection and no adaptive proposal covariance updating as in DRAM.

Default Behavior

Five MCMC algorithm variants are supported currently in QUESO: `dram`, `delayed_rejection`, `adaptive_metropolis`, `metropolis_hastings`, and `multilevel`. The default is `dram`.

Four MCMC algorithm variants are currently supported in MUQ: `dram`, `delayed_rejection`, `metropolis_hastings` and `adaptive_metropolis`.

Usage Tips

If the user wants to use Metropolis-Hastings, possibly as a comparison to the other methods which involve more chain adaptation, this is the MCMC type to use.

Examples

```
method,
  bayes_calibration queso
  metropolis_hastings
  samples = 10000 seed = 348
```

7.2.68.5.8 proposal_covariance

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)

Defines the technique used to generate the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Proposal Covariance Source (Group 1)	prior	Uses the covariance of the prior distributions to define the MCMC proposal covariance.
			derivatives	Use derivatives to inform the MCMC proposal covariance.
			values	Specifies matrix values to use as the MCMC proposal covariance.
			filename	Uses a file to import a user-specified MCMC proposal covariance.

Description

The proposal covariance is used to define a multivariate normal (MVN) jumping distribution used to create new points within a Markov chain. That is, a new point in the chain is determined by sampling within a MVN probability density with prescribed covariance that is centered at the current chain point. The accuracy of the proposal covariance has a significant effect on rejection rates and the efficiency of chain mixing.

Default Behavior

The default proposal covariance is `prior` when no emulator is present; `derivatives` when an emulator is present.

Expected Output

The effect of the proposal covariance is reflected in the MCMC chain values and the rejection rates, which can be seen in the diagnostic outputs from the QUESO solver within the `QuesoDiagnostics` directory.

Usage Tips

When derivative information is available inexpensively (e.g., from an emulator model), the derived-based proposal covariance forms a more accurate proposal distribution, resulting in lower rejection rates and faster chain mixing.

7.2.68.5.8.1 `prior`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [prior](#)

Uses the covariance of the prior distributions to define the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		multiplier	Multiplier to scale prior variance

Description

This keyword selection results in definition of the MCMC proposal covariance from the covariance of the prior distributions. This covariance is currently assumed to be diagonal without correlation.

Default Behavior

This is the default `proposal_covariance` option.

Usage Tips

Since this proposal covariance is defined globally, the chain does not need to be periodically restarted using local updates to this proposal. However, it is usually effective to adapt the proposal using one of the adaptive metropolis MCMC options.

Examples

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  dram
  proposal_covariance prior
```

7.2.68.5.8.2 multiplier

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [prior](#)
- [multiplier](#)

Multiplier to scale prior variance

Specification

Alias: none

Argument(s): REAL

Default: 1.0

Description

The initial proposal covariance will be given by the prior variance times the multiplier.

Default Behavior When using prior-based proposal covariance, the default is to use the prior variance for the proposal (multiplier = 1.0)

Usage Tips

The prior variance may result in too tight or narrow a proposal covariance. The multiplier can be used to scale all entries of the prior variance in determining the initial proposal covariance.

Examples

```
method
  bayes_calibration queso
    samples = 2000 seed = 348
  dram
    proposal_covariance prior
    multiplier 0.1
```

7.2.68.5.8.3 derivatives

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [derivatives](#)

Use derivatives to inform the MCMC proposal covariance.

Topics

This keyword is related to the topics:

- [bayesian_calibration](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		update_period	Period at which to update derivative-based proposal covariance

Description

This keyword selection results in definition of the MCMC proposal covariance from the Hessian of the misfit function (negative log likelihood), where this Hessian is defined from either a Gauss-Newton approximation (using only first derivatives of the calibration terms) or a full Hessian (using values, first derivatives, and second derivatives of the calibration terms). If this Hessian is indeterminate, it will be corrected as described in [16]

Default Behavior The default is `prior` based proposal covariance. This is a more advanced option that exploits structure in the form of the likelihood.

Expected Output

When derivatives are specified for defining the proposal covariance, the misfit Hessian and its inverse (the MVN proposal covariance) will be output to the standard output stream.

Usage Tips

The full Hessian of the misfit is used when either supported by the emulator in use (for PCE and surfpack GP, but not SC or dakota GP) or by the user's response specification (Hessian type is not "no_hessians"), in the case of no emulator. When this full Hessian is indefinite and cannot be inverted to form the proposal covariance, fallback to the positive semi-definite Gauss-Newton Hessian is employed.

Since this proposal covariance is locally accurate, it should be updated periodically using the `update_period` option. While the adaptive metropolis option can be used in combination with derivative-based preconditioning, it is generally preferable to instead decrease the proposal update period due to the improved local accuracy of this approach.

Examples

Generate a 2000 sample posterior chain, using derivatives to initialize the proposal covariance at the start of the chain.

```
method,
  bayes_calibration queso
  samples = 2000 seed = 348
  delayed_rejection
  emulator pce sparse_grid_level = 2
  proposal_covariance derivatives
```

7.2.68.5.8.4 update_period

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [derivatives](#)
- [update_period](#)

Period at which to update derivative-based proposal covariance

Specification

Alias: none

Argument(s): INTEGER

Description

For derivative-based proposal covariance, this specifies the period (number of accepted MCMC samples) after which the proposal covariance is updated using derivative values at the current chain point.

Default Behavior

When `update_period` is not specified, derivatives will inform the proposal covariance at the start of the chain, but not updated further.

Usage Tips

The `update_period` should be tailored to the size of the total chain, accounting for the relative expense of derivative-based proposal updates.

Examples

```
method,
  bayes_calibration queso
    samples = 2000 seed = 348
    delayed_rejection
    emulator pce sparse_grid_level = 2
    proposal_covariance derivatives
      update_period = 40 # update proposal covariance every 40 points
```

7.2.68.5.8.5 values

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [values](#)

Specifies matrix values to use as the MCMC proposal covariance.

Specification

Alias: none

Argument(s): REALLIST

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Values For (Group 1)	Dakota Keyword diagonal	Dakota Keyword Description Specifies the diagonal matrix format when specifying a user-specified proposal covariance.
			matrix	Specifies the full matrix format when specifying a user-specified proposal covariance.

Description

This keyword selection results in definition of the MCMC proposal covariance from user-specified matrix values. The matrix input format must be declared as either a full matrix or a matrix diagonal.

Default Behavior

This option is not the default, and generally implies special a priori knowledge from the user.

Usage Tips

This option is not supported for the case of transformations to standardized probability space.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  values ... # See leaf nodes for required format option
```

7.2.68.5.8.6 diagonal

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [values](#)
- [diagonal](#)

Specifies the diagonal matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    diagonal values 1.0e6 1.0e-1
```

7.2.68.5.8.7 matrix

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [values](#)
- [matrix](#)

Specifies the full matrix format when specifying a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When specifying the MCMC proposal covariance in an input file, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
    matrix values 1.0 0.1
                 0.1 2.0
```

7.2.68.5.8.8 filename

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [muq](#)
- [proposal_covariance](#)
- [filename](#)

Uses a file to import a user-specified MCMC proposal covariance.

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Values For (Group 1)	Dakota Keyword diagonal	Dakota Keyword Description Specifies the diagonal matrix format when importing a user-specified proposal covariance.
			matrix	Specifies the full matrix format when importing a user-specified proposal covariance.

Description

This keyword selection results in definition of the MCMC proposal covariance from importing data a user-specified filename. This import must be declared as either a full matrix or a matrix diagonal.

Default Behavior

This option is not the default, and generally implies special a priori knowledge from the user.

Usage Tips

This option is not supported for the case of transformations to standardized probability space.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  filename ... # See leaf nodes for required format option
```

7.2.68.5.8.9 diagonal

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [filename](#)
- [diagonal](#)

Specifies the diagonal matrix format when importing a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a diagonal matrix format, i.e., the user only provides the (positive) values along the diagonal.

Examples

```
method,  
    bayes_calibration queso  
    samples = 1000 seed = 348  
    dram  
    proposal_covariance  
    diagonal filename 'dakota_cantilever_queso.diag.dat'
```

7.2.68.5.8.10 matrix

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [muq](#)
- [proposal_covariance](#)
- [filename](#)
- [matrix](#)

Specifies the full matrix format when importing a user-specified proposal covariance.

Specification

Alias: none

Argument(s): none

Description

When importing the MCMC proposal covariance from a user-specified filename, this keyword declares the use of a full matrix format, i.e., the user provides all values of the matrix, not just the diagonal. The matrix must be symmetric, positive-definite.

Examples

```
method,
  bayes_calibration queso
  samples = 1000 seed = 348
  dram
  proposal_covariance
  matrix filename 'dakota_cantilever_queso.matrix.dat'
```

7.2.68.6 experimental_design

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)

(Experimental) Adaptively select experimental designs for iterative Bayesian updating

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		initial_samples	Number of data points used during initial Bayesian calibration
	Required		num_candidates	Number of candidate design points considered
	Optional		max_hifi_evaluations	Maximum number of high-fidelity model runs to be used
	Optional		batch_size	Number of optimal designs selected concurrently
	Optional		import_candidate_points_file	Specify text file containing candidate design points

	Optional	ksg2	Use second Kraskov algorithm to compute mutual information
--	-----------------	----------------------	--

Description

This "experimental design" algorithm uses responses produced by a high-fidelity model as data to update the parameters of a low-fidelity model using Bayes' Rule. It is a capability that is under active development and is currently compatible only with `queso`. The user-specified high-fidelity model should depend only on configuration variables (i.e. design conditions), such as temperature or spatial location, while the user-specified low-fidelity model should depend on both configuration variables and its own model parameters to be calibrated.

The algorithm starts with a preliminary Bayesian calibration using the number of data points specified in `initial_samples`. These data points can be read in through the `calibration_data_file` command in the `responses` block. If `num_experiments` is less than `initial_samples` or if no such data file is provided, Latin Hypercube Samples of the design space (specified in the `variables` block) will be run through the user-specified high-fidelity code to supplement the initial data. Once this first calibration is complete, a set of possible experimental design conditions, specified using the configuration variables, is proposed. The user specifies the size of this set using `num_candidates`. The set of candidates itself may be explicitly given through the `import_candidate_points_file` command. If the number of candidates in this file is less than `num_candidates`, or if this file is omitted, the set of candidate designs will again be supplemented with a Latin Hypercube Sample of the design space.

For each candidate design ξ_i in the set of possible design conditions, the mutual information (MI) between the low-fidelity model parameters θ and the high-fidelity model response $\mathbf{y}(\xi_i)$,

$$MI = \iint f(\theta, \mathbf{y}(\xi_i)) \log \frac{f(\theta, \mathbf{y}(\xi_i))}{f(\theta)f(\mathbf{y}(\xi_i))} d\theta d\mathbf{y},$$

is approximated. The high-fidelity model is replaced by the low-fidelity model and a k -nearest neighbor approximation is used in the calculation. The design point ξ^* for which MI is the largest is selected and run through the high-fidelity model to yield a new observation $\mathbf{y}(\xi^*)$. This new observation is added to the calibration data, and a subsequent Bayesian calibration is performed. A new MI for each remaining candidate design is computed, and the process repeats until one of three stopping criteria are met. Multiple optimal designs may be selected concurrently by specifying `batch_size`.

Of the three stopping criteria, two are automatically checked by Dakota. If the relative change in the MI from one iteration to the next is sufficiently small or if the set of candidate design conditions has been exhausted, the algorithm terminates. The user may specify the third stopping criteria using `max_hifi_evaluations`. This limits the number of high-fidelity model evaluations that will be performed during this algorithm. It therefore limits the number of iterations through the algorithm that will be performed. Any high-fidelity model runs needed to produce the data set for the initial calibration are not included in this allocation.

In the case that the high-fidelity model must be run independently of Dakota, the user may set `max_hifi_evaluations` to zero. The optimal experimental design point will be calculated and reported, but the high-fidelity model will not be run. For more details, see the User's Manual.

Expected Output

Information regarding the progress and termination condition of the experimental design algorithm is output to the screen with varying levels of verbosity. Further details can be found, regardless of verbosity, in the output file `experimental_design_output.txt`

Usage Tips

Due to the optional file read-ins and the supplemental sampling, it is important for the user to check consistency within the input file specifications. For example, if `num_experiments` is less than the number of experiments in the `calibration_data_file`, only the first lines of the file will be used and the rest will be discarded. The same holds true for the `import_candidate_points_file` and `num_candidates`.

Examples

The input file below illustrates the use of `experimental_design` and its options

```
bayes_calibration
  queso
  dram
  chain_samples = 1000 seed = 348
experimental_design
  initial_samples = 5
  num_candidates = 10
  import_candidate_points_file = "bayes_experimental_design.cand.dat"
  freeform
  max_hifi_evaluations = 3
```

7.2.68.6.1 initial_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [initial_samples](#)

Number of data points used during initial Bayesian calibration

Specification

Alias: samples

Argument(s): INTEGER

Default: method-dependent

Description

The experimental design algorithm starts with a preliminary Bayesian calibration. The keyword `initial_samples` specifies the number of data points used during this initial calibration. These data points may come from external data sources through `calibration_data_file`, from an LHS sample of the design space, or both.

7.2.68.6.2 num_candidates

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [num_candidates](#)

Number of candidate design points considered

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

After the initial Bayesian calibration, a set of candidate design points is created, from which the optimal design(s) will be chosen. This set is of size `num_candidates`. These candidate points may be specified by the user through `import_candidate_points_file`, from an LHS sample of the design space, or both.

7.2.68.6.3 `max_hifi_evaluations`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [max_hifi_evaluations](#)

Maximum number of high-fidelity model runs to be used

Specification

Alias: none

Argument(s): INTEGER

Description

There are three stopping criteria by which the Bayesian experimental design algorithm will be terminated:

- The high-fidelity model has been run `max_hifi_evaluations` times
- The set of candidate design points have been exhausted
- The relative change in the mutual information from one iteration to the next is sufficiently small The first criterion is controlled by this keyword specification

Default Behavior

If no maximum number of high-fidelity model runs is specified, only the last two stopping criteria listed above will be evaluated.

Usage Tips

Be wary that `max_hifi_evaluations` does not include any high-fidelity model evaluations that need to be run to produce the data set for the initial Bayesian calibration. This number only includes those evaluations performed on the sequentially chosen optimal design points

If `max_hifi_evaluations` is set to zero, the a single optimal experimental design point will be selected, and Dakota will exit without running the high- fidelity model.

7.2.68.6.4 `batch_size`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [batch_size](#)

Number of optimal designs selected concurrently

Specification

Alias: none

Argument(s): INTEGER

Description

Each iteration of the `experimental_design` algorithm yields `batch_size` optimal designs. If `max_hifi` is set to zero, these values will merely be reported in Dakota output and the file `experimental_design_output.txt`. Otherwise, the high-fidelity model will be run at all optimal designs, and the responses will be added to the list of experiment data before the Bayesian calibration is performed in the next iteration of the algorithm.

Default Behavior

By default, optimal designs will be selected one at a time.

7.2.68.6.5 `import_candidate_points_file`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [import_candidate_points_file](#)

Specify text file containing candidate design points

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Enables text file import of user-specified design points (i.e. configuration variables). No model responses should be included. Each row should correspond to a single design or configuration.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.68.6.5.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [import_candidate_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.68.6.5.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [import_candidate_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.6.5.3 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [import_candidate_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.6.5.4 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [import_candidate_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent [custom_annotated](#)

7.2.68.6.5.5 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [import_candidate_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.6.5.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [import_candidate_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.6.6 ksg2

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [experimental_design](#)
- [ksg2](#)

Use second Kraskov algorithm to compute mutual information

Specification

Alias: none

Argument(s): none

Description

This algorithm is derived in [58]. The mutual information between m random variables is approximated by

$$I_2(X_1, X_2, \dots, X_m) = \psi(k) + (m-1)\psi(N) - (m-1)/k - \langle \psi(n_{x_1}) + \psi(n_{x_2}) + \dots + \psi(n_{x_m}) \rangle,$$

where ψ is the digamma function, k is the number of nearest neighbors being used, and N is the number of samples available for the joint distribution of the random variables. For each point $z_i = (x_{1,i}, x_{2,i}, \dots, x_{m,i})$ in the joint distribution, z_i and its k nearest neighbors are projected into each marginal subspace. For each subspace $j = 1, \dots, m$, $\varepsilon_{j,i}$ is defined as the radius of the l_∞ -ball containing all $k+1$ points. Then, $n_{x_{j,i}}$ is the number of points in the j -th subspace within a distance of $\varepsilon_{j,i}$ from the point $x_{j,i}$. The angular brackets denote that the average of $\psi(n_{x_{j,i}})$ is taken over all points $i = 1, \dots, N$.

Examples

```
method
  bayes_calibration queso
  dram
  seed = 34785
  chain_samples = 1000
  posterior_stats mutual_info
  ksg2
```

```
method
  bayes_calibration
  queso
  dram
  chain_samples = 1000 seed = 348
  experimental_design
  initial_samples = 5
  num_candidates = 10
  max_hifi_evaluations = 3
  ksg2
```

7.2.68.7 calibrate_error_multipliers

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [calibrate_error_multipliers](#)

Calibrate hyper-parameter multipliers on the observation error covariance

Specification

Alias: none

Argument(s): none

Default: none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Calibrate Error Multipliers (Group 1)	one	Calibrate one hyper-parameter multiplier across all re-sponses/experiments
			per_experiment	Calibrate one hyper-parameter multiplier per experiment
			per_response	Calibrate one hyper-parameter multiplier per response
			both	Calibrate one hyper-parameter multiplier for each re-sponse/experiment pair
	Optional		hyperprior_alphas	Shape (alpha) of the inverse gamma hyper-parameter prior

Description

Calibrate one or more multipliers on the user-provided observation error covariance ([experiment_variance_type](#)). Options include [one](#) multiplier on the whole block-diagonal covariance structure, one multiplier [per_experiment](#) covariance block, one multiplier [per_response](#) covariance block, or separate multipliers for each response/experiment pair (for a total of number experiments X number response groups).

Default Behavior: No hyper-parameter calibration. When hyper-parameter calibration is enabled, the default prior on the multiplier is a diffuse inverse gamma, with mean and mode approximately 1.0.

Expected Output: Final calibration results will include both inference parameters and one or more calibrated hyper-parameters.

Usage Tips: The [per_response](#) option can be useful when each response has its own measurement error process, but all experiments were gathered with the same equipment and conditions. The [per_experiment](#) option might be used when working with data from multiple independent laboratories.

Examples

Perform Bayesian calibration with 2 calibration variables and two hyper-parameter multipliers, one per each of two responses. The multipliers are assumed the same across the 10 experiments. The priors on the multipliers are specified using the [hyperprior_alphas](#) and [hyperprior_betas](#) keywords.

```

bayes_calibration queso
  samples = 1000 seed = 348
  dram
  calibrate_error_multipliers per_response
  hyperprior_alphas = 27.0

```

```
hyperprior_betas = 26.0

variables
  uniform_uncertain 2
  upper_bounds 1.e8 10.0
  lower_bounds 1.e6 0.1
  initial_point 2.85e7 2.5
  descriptors 'E' 'w'

responses
  calibration_terms = 2
  calibration_data_file = 'expdata.withsigma.dat'
  freeform
  num_experiments = 10
  experiment_variance_type = 'scalar'
```

7.2.68.7.1 one

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [calibrate_error_multipliers](#)
- [one](#)

Calibrate one hyper-parameter multiplier across all responses/experiments

Specification

Alias: none

Argument(s): none

Description

A single hyper-parameter multiplying all response/experiment covariance blocks will be estimated.

7.2.68.7.2 per_experiment

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [calibrate_error_multipliers](#)
- [per_experiment](#)

Calibrate one hyper-parameter multiplier per experiment

Specification

Alias: none

Argument(s): none

Description

One hyper-parameter multiplying each experiment covariance block will be estimated.

7.2.68.7.3 per_response

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [calibrate_error_multipliers](#)
- [per_response](#)

Calibrate one hyper-parameter multiplier per response

Specification

Alias: none

Argument(s): none

Description

One hyper-parameter multiplying each response covariance block will be estimated.

7.2.68.7.4 both

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [calibrate_error_multipliers](#)
- [both](#)

Calibrate one hyper-parameter multiplier for each response/experiment pair

Specification

Alias: none

Argument(s): none

Description

One hyper-parameter multiplying each experiment/response covariance block will be estimated.

7.2.68.7.5 hyperprior_alphas

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)

- [calibrate_error_multipliers](#)
- [hyperprior_alphas](#)

Shape (alpha) of the inverse gamma hyper-parameter prior

Specification

Alias: none

Argument(s): REALLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		hyperprior_betas	Scale (beta) of the inverse gamma hyper-parameter prior

Description

Shape of the prior distribution for calibrated error multipliers. Either a single value or number of hyper-parameters values may be specified.

Default: 102.0 (with beta = 103.0) so mean and mode are approximately 1.0 and standard deviation is about 0.1.

7.2.68.7.5.1 [hyperprior_betas](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [calibrate_error_multipliers](#)
- [hyperprior_alphas](#)
- [hyperprior_betas](#)

Scale (beta) of the inverse gamma hyper-parameter prior

Specification

Alias: none

Argument(s): REALLIST

Description

Scale of the prior distribution for calibrated error multipliers. Either a single value or number of hyper-parameters values may be specified.

Defaults to 103.0 (with alpha = 102.0) so mean and mode are approximately 1.0 and standard deviation is about 0.1.

7.2.68.8 burn_in_samples

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [burn_in_samples](#)

Manually specify the burn in period for the MCMC chain.

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

The burn in period is the number of samples at the beginning of the MCMC chain that are discarded. The resulting chain is less dependent on the starting point of the chain.

Default Behavior

If not specified, no MCMC samples are discarded

Expected Output

If `burn_in_samples` is specified, an additional tabular output file containing the final MCMC chain is generated.

Examples

Generate a chain of length 1000 and discard the first 100 points, resulting in a chain of length 900.

```
method,  
  bayes_calibration queso  
    chain_samples = 1000 seed = 348  
    dram  
  proposal_covariance  
    diagonal values 1.0e6 1.0e-1  
  burn_in_samples 100
```

7.2.68.9 posterior_stats

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [posterior_stats](#)

Compute information-theoretic metrics on posterior parameter distribution

Specification**Alias:** none**Argument(s):** none**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		kl_divergence	Calculate the Kullback-Leibler Divergence between prior and posterior
	Optional		mutual_info	Calculate the mutual information between prior and posterior
	Optional		kde	Calculate the Kernel Density Estimate of the posterior distribution

Description

Information theory allows for the calculation and quantification of the information contained within a distribution. It is particularly useful for distributions which are not Gaussian, such as those which are bimodal or highly skewed. The `posterior_stats` command calculates information metrics relating to the posterior distribution of the model parameters. These metrics are approximated by making use of the MCMC chain produced during the Bayesian update. This capability can be used with any Bayesian method.

7.2.68.9.1 [kl_divergence](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [posterior_stats](#)
- [kl_divergence](#)

Calculate the Kullback-Leibler Divergence between prior and posterior

Specification

Alias: none

Argument(s): none

Description

The Kullback-Leibler (KL) Divergence, also called the relative entropy, provides a measure of the difference between two probability distributions. By specifying `kl_divergence`, the KL Divergence between the posterior $f(\theta|\mathbf{y}^{Data})$ and the prior $f(\theta)$ parameter distributions is calculated such that

$$D_{KL} = \int f(\theta|\mathbf{y}^{Data}) \log \frac{f(\theta|\mathbf{y}^{Data})}{f(\theta)} d\theta$$

This quantity can be interpreted as the amount of information gained about the parameters during the Bayesian update.

Expected Output

If `kl_divergence` is specified, the calculated value will be reported to the screen at the end of the calibration, following the sample statistics of the response functions. Example output is given below.

Additional Discussion

The quantity calculated is a k -nearest neighbor approximation of the possibly multi-dimensional integral given above. Therefore, some applications whose true KL Divergence is quite close to zero may report a negative KL Divergence.

Examples

Below is a `method` block of a Dakota input file that indicates the calculation of the KL Divergence

```
method,
  bayes_calibration queso
  dram
  seed = 34785
  chain_samples = 1000
  posterior_stats kl_divergence
```

The calculated KL approximation is indicated in the screen output by "Information gained from prior to posterior" as shown below

```
Sample moment statistics for each response function:
              Mean              Std Dev              Skewness              Kurtosis
least_sq_term_1  3.9982462078e-01  4.7683816550e-04  -2.3448518080e+00  7.7381497770e+00

Information gained from prior to posterior = 1.0066819600e+01

<<<<< Iterator bayes_calibration completed.
<<<<< Environment execution completed.
```

7.2.68.9.2 mutual_info

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [posterior_stats](#)
- [mutual_info](#)

Calculate the mutual information between prior and posterior

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	ksg2	Use second Kraskov algorithm to compute mutual information
--	-----------------	----------------------	--

Description

The mutual information quantifies how much information two random variables contain about each other. It is a measure of the mutual dependence of two random variables. The mutual information is a non-negative measure, with zero representing complete independence of the two random variables. For continuous random variables X and Y , the mutual information is

$$I(X,Y) = \int \int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dx dy.$$

The mutual information can also be interpreted as the reduction in uncertainty of one random variable due to the knowledge of another. By specifying, `mutual_info`, the mutual information between the posterior parameters and the prior parameters is calculated.

The mutual information is calculated using a k -nearest neighbor approximation algorithm. As of Dakota 6.6, there are two such algorithms available, both of which are derived in [58]. By default, Dakota uses the first such algorithm; the second may be selected by specifying the keyword `ksg2`. Further details can be found in the Dakota Theory Manual [16].

Expected Output

If `mutual_information` is specified, the calculated value will be reported to the screen at the end of the calibration.

Additional Discussion

Due to the necessary approximation of the multidimensional integral above, a negative mutual information may be reported for applications whose true value is close to or equal to zero. As of Dakota 6.6, mutual information calculations are primarily used in the implementation of the [experimental_design](#) algorithm.

Examples

```
method
  bayes_calibration queso
  dram
  seed = 34785
  chain_samples = 1000
  posterior_stats mutual_info
```

7.2.68.9.2.1 ksg2

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [posterior_stats](#)
- [mutual_info](#)
- [ksg2](#)

Use second Kraskov algorithm to compute mutual information

Specification

Alias: none

Argument(s): none

Description

This algorithm is derived in [58]. The mutual information between m random variables is approximated by

$$I_2(X_1, X_2, \dots, X_m) = \psi(k) + (m-1)\psi(N) - (m-1)/k - \langle \psi(n_{x_1}) + \psi(n_{x_2}) + \dots + \psi(n_{x_m}) \rangle,$$

where ψ is the digamma function, k is the number of nearest neighbors being used, and N is the number of samples available for the joint distribution of the random variables. For each point $z_i = (x_{1,i}, x_{2,i}, \dots, x_{m,i})$ in the joint distribution, z_i and its k nearest neighbors are projected into each marginal subspace. For each subspace $j = 1, \dots, m$, $\varepsilon_{j,i}$ is defined as the radius of the l_∞ -ball containing all $k+1$ points. Then, $n_{x_{j,i}}$ is the number of points in the j -th subspace within a distance of $\varepsilon_{j,i}$ from the point $x_{j,i}$. The angular brackets denote that the average of $\psi(n_{x_{j,i}})$ is taken over all points $i = 1, \dots, N$.

Examples

```
method
  bayes_calibration queso
  dram
  seed = 34785
  chain_samples = 1000
  posterior_stats mutual_info
  ksg2
```

```
method
  bayes_calibration
  queso
  dram
  chain_samples = 1000 seed = 348
  experimental_design
  initial_samples = 5
  num_candidates = 10
  max_hifi_evaluations = 3
  ksg2
```

7.2.68.9.3 kde

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [posterior_stats](#)
- [kde](#)

Calculate the Kernel Density Estimate of the posterior distribution

Specification

Alias: none

Argument(s): none

Description

A kernel density estimate (KDE) is a non-parametric, smooth approximation of the probability density function of a random variable. It is calculated using a set of samples of the random variable. If X is a univariate random variable with unknown density f and independent and identically distributed samples x_1, x_2, \dots, x_n , the KDE is given by

$$\hat{f} = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right).$$

The kernel K is a non-negative function which integrates to one. Although the kernel can take many forms, such as uniform or triangular, Dakota uses a normal kernel. The bandwidth h is a smoothing parameter that should be optimized. Choosing a large value of h yields a wide KDE with large variance, while choosing a small value of h yields a choppy KDE with large bias. Dakota approximates the bandwidth using Silverman's rule of thumb,

$$h = \hat{\sigma} \left(\frac{4}{3n} \right)^{1/5},$$

where $\hat{\sigma}$ is the standard deviation of the sample set $\{x_i\}$.

For multivariate cases, the random variables are treated as independent, and a separate KDE is calculated for each.

Expected Output

If `kde` is specified, calculated values of \hat{f} will be output to the file `kde_posterior.dat`. Example output is given below.

Examples

Below is a `method` block of a Dakota input file that indicates the calculation of the KDE

```
method,
  bayes_calibration queso
  dram
  seed = 34785
  chain_samples = 1000
  posterior_stats kde
```

The calculated KDE values are output to the file `kde_posterior.dat`, as shown below

```
uuv_1 KDE PDF estimate
0.406479 61.2326
0.406338 64.0245
0.402613 114.468
0.402613 114.468
0.40249 114.409
0.40282 114.162
0.398899 65.2361
0.400093 84.9285
0.401264 104.105
0.400917 98.7803
```

7.2.68.10 chain_diagnostics

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [chain_diagnostics](#)

Compute diagnostic metrics for Markov chain

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		confidence_intervals	Calculate the confidence intervals on estimates of first and second moments

Description

While a Markov chain produced via Monte Carlo sampling eventually converges to a set of samples representative of an underlying probability distribution, the first set of samples may not accurately capture the target distribution. Chain diagnostic metrics provide measures of this convergence, and are intended to help users decide whether `samples` should be increased in Bayesian calibration exercises.

7.2.68.10.1 [confidence_intervals](#)

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [chain_diagnostics](#)
- [confidence_intervals](#)

Calculate the confidence intervals on estimates of first and second moments

Specification

Alias: none

Argument(s): none

Description

During Bayesian calibration, a chain of samples is produced, which represents the underlying posterior distribution of model parameters. For each parameter sample, the corresponding model response is computed. The `confidence_intervals` keyword indicates the calculation of a 95% confidence interval on the estimated mean and variance of each parameter and each response.

As of Dakota 6.10, these confidence intervals are calculated using the asymptotically valid interval estimator,

$$\bar{g}_n \pm t_* \frac{\hat{\sigma}_n}{\sqrt{n}},$$

where \bar{g}_n is the moment (i.e. mean or variance), t_* is the Student's t -value for the 95th quantile, n is the sample size, and $\hat{\sigma}_n$ is an estimate of the standard error whose square is obtained using batch means estimation. The Markov

chain produced during calibration is broken up into "batches," the sample moment is calculated for each batch, and $\hat{\sigma}_n$ is an unbiased estimate of the standard deviation in these batch moment calculations.

Expected Output

If `confidence_intervals` is specified, the 95% confidence interval for the mean and variance for each parameter and for each response will be output to the screen. If `output` is set to `debug`, the mean of the moment calculated for each batch will also be output to the screen.

Additional Discussion

Confidence intervals may be used to indicate to the user whether `samples` needs to be increased during the Bayesian calibration. For example, if the width of the intervals (one, many, or all) is below some threshold value, that may indicate that enough samples have been drawn.

Examples

Below is a `method` block of a Dakota input file that indicates the calculation of confidence intervals

```
method,
  bayes_calibration queso
  dram
  seed = 34785
  chain_samples = 1000
  chain_diagnostics
  confidence_intervals
```

The calculated confidence intervals are output to the screen under "Chain diagnostics":

```
Sample moment statistics for each posterior variable:
      Mean          Std Dev      Skewness      Kurtosis
E  2.8609959149e+07  1.4417714265e+05  8.0289072766e-01  7.8655956160e-02
w  2.5016445558e+00  3.8306697138e-03  -1.2217188066e-01  3.8866929786e-02

Sample moment statistics for each response function:
      Mean          Std Dev      Skewness      Kurtosis
stress  2.6282814617e+03  8.9765361327e+01  1.3400226598e-01  4.9239052296e-02
displacement  2.9604502307e-01  1.0636886950e-02  -3.5080744509e-01  -1.2381836901e-01

Chain diagnostics
95% Confidence Intervals of means
      E = [2.8570364780e+07, 2.8649553519e+07]
      w = [2.5009524557e+00, 2.5023366559e+00]
      stress = [2.6120609285e+03, 2.6445019948e+03]
      displacement = [2.9337418438e-01, 2.9871586175e-01]
95% Confidence Intervals of variances
      E = [1.5074828429e+10, 2.6499268497e+10]
      w = [1.1359880885e-05, 1.7988180028e-05]
      stress = [6.2340446164e+03, 9.8815955721e+03]
      displacement = [8.8187472572e-05, 1.3809925539e-04]
```

7.2.68.11 model_evidence

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_evidence](#)

Calculate model evidence (marginal likelihood of model) when using Bayesian methods

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mc_approx	Calculate model evidence using a Monte Carlo sampling approach
	Optional		evidence_samples	The number of samples used in Monte Carlo approximation of the model evidence.
	Optional		laplace_approx	Calculate model evidence using the Laplace approximation

Description

Model evidence is used in Bayesian model selection and model averaging. It is defined as the probability of the data given the model, and is calculated by averaging the likelihood of the model parameters over all values of the model parameters according to their prior distributions. In Dakota, one must calculate the model evidence separately for each model and perform the normalization to obtain the posterior model plausibility for each model.

Default Behavior

When specifying `model_evidence`, there are two methods of calculating it. One or both may be specified. They include the Monte Carlo approximation, given by `mc_approx` and the Laplace approximation, given by `laplace_approx`. `mc_approx` is the default approach.

Expected Output Currently, the model evidence will be printed in the screen output with prefacing text indicating if it is calculated by Monte Carlo sampling or the Laplace approximation.

Usage Tips

7.2.68.11.1 `mc_approx`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_evidence](#)
- [mc_approx](#)

Calculate model evidence using a Monte Carlo sampling approach

Specification

Alias: none

Argument(s): none

Description

The `mc_approx` keyword for model evidence indicates that sample values will be generated from the prior distribution, and then the simulation model will be evaluated at these sample values to obtain corresponding likelihood values. The average of the likelihood weighted by the prior is the model evidence. The accuracy of this approximation depends on the number of samples taken, which is specified by the `evidence_samples` keyword.

Default Behavior

If `evidence_samples` is not specified with `mc_approx`, Dakota uses the number of chain samples from the MCMC (`chain_samples`) as the number of samples to use for calculating the model evidence.

Expected Output Currently, the model evidence will be printed in the screen output with prefacing text indicating if it is calculated by Monte Carlo sampling.

Usage Tips

7.2.68.11.2 `evidence_samples`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_evidence](#)
- [evidence_samples](#)

The number of samples used in Monte Carlo approximation of the model evidence.

Specification

Alias: none

Argument(s): INTEGER

Description

The `mc_approx` keyword for model evidence indicates that sample values will be generated from the prior distribution, and then the simulation model will be evaluated at these sample values to obtain corresponding likelihood values. The average of the likelihood weighted by the prior is the model evidence. The accuracy of this approximation depends on the number of samples taken, which is specified by the `evidence_samples` keyword. Note that each sample specified by the `evidence_samples` keyword will require an evaluation of the simulation model to compute the corresponding likelihood. So, this may become costly for expensive simulations. Additionally, many prior samples will have very low (near zero) likelihood, so millions of samples may be required for accurate computation of the integral which defines model evidence.

Default Behavior

If `evidence_samples` is not specified with `mc_approx`, Dakota uses the number of chain samples from the MCMC (`chain_samples`) as the number of samples to use for calculating the model evidence.

Expected Output Currently, the model evidence will be printed in the screen output with prefacing text indicating if it is calculated by Monte Carlo sampling.

Usage Tips

7.2.68.11.3 `laplace_approx`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_evidence](#)
- [laplace_approx](#)

Calculate model evidence using the Laplace approximation

Specification

Alias: none

Argument(s): none

Description

The `laplace_approx` keyword for model evidence indicates that a pre-solve will be used prior to the Bayesian MCMC sampling to estimate the Maximum A Posteriori (MAP) point. The Laplace approximation assumes the posterior density is nearly Gaussian and is given by a formula which involves the likelihood at the MAP point, the prior density at the MAP point, and the Hessian of the log-posterior at the MAP point. The formula is given in the Dakota User's manual. This method is efficient at estimating the model evidence for posterior densities with weak non-Gaussian characteristics but it does require a MAP solve (so `pre-solve` should be specified) and it does require gradient and Hessians of the response to be on.

Default Behavior

Expected Output Currently, the model evidence will be printed in the screen output with prefacing text indicating if it is calculated by the Laplace approximation.

Usage Tips

7.2.68.12 `model_discrepancy`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)

(Experimental) Post-calibration calculation of model discrepancy correction

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		discrepancy_type	Specify the type of model discrepancy
	Optional		num_prediction_configs	Specify number of prediction locations for model discrepancy
	Optional		prediction_configs	List prediction locations for model discrepancy
	Optional		import_prediction_configs	Specify text file containing prediction configurations for model discrepancy
	Optional		export_discrepancy_file	Output file for prediction discrepancy calculations
	Optional		export_corrected_model_file	Output file for corrected model prediction calculations
	Optional		export_corrected_variance_file	Output file for prediction variance calculations

Description

The goal of parameter calibration is to minimize the difference between experimental data, $d(x)$, and model observations, $M(\theta, x)$, where θ are the model parameters and x is a configuration variable, such as temperature or pressure. However, it is not uncommon that, at the conclusion of parameter calibration, the agreement between experimental data and the calibrated model is not "close enough." This is often due to model form or structural error. In this case, the canonical equation

$$d(x) = M(\theta, x) + \varepsilon$$

is replaced by one that also includes model discrepancy $\delta(x)$,

$$d(x) = M(\theta, x) + \delta(x) + \varepsilon.$$

In the Dakota implementation, the calculation of $\delta(x)$ is performed after the model parameters θ are calibrated. For each observable d_i , the discrepancy

$$\delta_i(x_j) = d_i(x_j) - M_i(\theta^*, x_j)$$

is calculated for each value x_j of the configuration variable, where θ^* is the optimal parameter value obtained during the calibration. For scalar responses, the model discrepancy is only a function of the configuration variables, and

there is one discrepancy regression model for each observable d_i . This set of discrepancy models may be specified to be either Gaussian process or polynomial regression models of constant, linear, or quadratic order, and each model is fit to the calculated discrepancy values. See the [discrepancy_type](#) command for more details regarding these options. For field responses, the model discrepancy is a function of the configuration variables as well as the independent field coordinates (such as time or space), and there is one discrepancy regression model for each field. In this case, the discrepancy models are Gaussian process models. The calculation of model discrepancy has not been tested for cases in which responses are mixed scalar and field responses.

The user may then specify new or "prediction" configurations at which the corrected model $M(\theta^*, x_{new}) + \delta(x_{new})$ should be calculated, using one of the `num_prediction_configs`, `prediction_configs`, or `import_prediction_configs` keywords. If none of these keywords is specified, the number of prediction configurations is set to 10 for scalar responses. The corresponding prediction variances are also calculated, according to

$$\Sigma_{total}(x) = \Sigma_{\delta}(x) + \sigma_{exp}^2 I.$$

Here, $\Sigma_{\delta}(x)$ is the (co)variance calculated from the Gaussian process or polynomial regression model, and σ_{exp}^2 is the maximum variance, taken over all configuration variables for that observation. In the case of field responses, the default prediction configurations are set equal to the input configurations, and the variance information contains only the variance calculated from the Gaussian process correction model. Further details can be found in the Dakota User's and Theory Manuals.

Usage Tips

For field responses, the keyword [read_field_coordinates](#) *must* be specified when computing the model discrepancy. See [field_calibration_terms](#) for more information regarding options for calibration with field responses.

Expected Output

The resulting values of $\delta(x_{new})$ will be exported to the file specified using `export_discrepancy_file` or to the default file `dakota_discrepancy_tabular.dat`. The values of the corrected model at the specified prediction configurations will be exported to the file specified using `export_corrected_model_file` or to the default file `dakota_corrected_model_tabular.dat`, and the corresponding prediction variances will be exported to `dakota_discrepancy_variance_tabular.dat` or the file specified with `export_corrected_variance_file`.

Examples

Extensive examples can be found in the Dakota User's and Theory Manuals. The input files below illustrate the options available to `model_discrepancy`

```
model_discrepancy
  discrepancy_type gaussian_process
  num_prediction_configs 11
  export_discrepancy_file "discrepancy_values.txt"
  export_corrected_model_file "corrected_model.txt"
  export_corrected_variance_file "prediction_variance.txt"
```

```
model_discrepancy
  discrepancy_type polynomial
  correction_order constant
  prediction_configs 1 1.5 2 2.5 3
```

```
model_discrepancy
  discrepancy_type polynomial
  correction_order linear
  import_prediction_configs "prediction_configs.txt"
```

7.2.68.12.1 discrepancy_type

- [Keywords Area](#)

- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [discrepancy_type](#)

Specify the type of model discrepancy

Specification

Alias: none

Argument(s): none

Default: gaussian process

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Discrepancy Model (Group 1)	Dakota Keyword gaussian_process	Dakota Keyword Description Use the Surfpack version of Gaussain process as the discrepancy model
			polynomial	Use a polynomial surrogate as the discrepancy model

Description

After the model parameters are calibrated, the difference between the data and the calibrated model, i.e. the model discrepancy, is calculated

$$\delta_i(x_j) = d_i(x_j) - M_i(\theta^*, x_j).$$

Each δ_i corresponds to a different regression model. These regression models must all be either Gaussian process or polynomial models, and they are functions of the configuration variable x . The order of the trend function may be selected using the `correction_order` command by specifying `constant`, `linear`, or `quadratic`.

Note that for Dakota 6.9 and earlier, this keyword only applies to discrepancy calculations for scalar responses. For field responses, a Gaussian process model with a quadratic function is used by default.

7.2.68.12.1.1 `gaussian_process`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [discrepancy_type](#)
- [gaussian_process](#)

Use the Surfpack version of Gaussain process as the discrepancy model

Specification

Alias: kriging

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		trend_order	Trend function order of the model discrepancy

Description

This keyword specifies a generic version of the Gaussian process implemented in the surface-fitting library called Surfpack. With the exception of the `correction_order` specification, all default options are used in the approximation of the model discrepancy function. That is, DIRECT is used as the optimization method for finding the maximum likelihood estimates of the hyper-parameters that govern the trend and correlation functions, the correlation lengths are optimized by Surfpack, and ill-conditioning is handled internally; gradient-enhanced kriging is not provided as an option in this application.

By selecting this option, each discrepancy function will take the form of a Gaussian process

7.2.68.12.1.2 `trend_order`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [discrepancy_type](#)
- [gaussian_process](#)
- [trend_order](#)

Trend function order of the model discrepancy

Specification

Alias: none

Argument(s): INTEGER

Default: 2

Description

The order of the trend function may be specified using the `trend_order` command, followed by either 0, for which a constant trend function will be used, 1, for which a linear trend function will be used, or 2, for which a quadratic trend function will be used.

Default Behavior

By default, a quadratic trend function is used.

7.2.68.12.1.3 polynomial

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [discrepancy_type](#)
- [polynomial](#)

Use a polynomial surrogate as the discrepancy model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		basis_order	Basis function order of the model discrepancy

Description

For the purposes of model discrepancy, constant, linear, and quadratic polynomial surrogate models are available. These models are less expensive to calculate than Gaussian process models but are consequently less flexible to discrepancy trends that are not constant, linear, or quadratic in nature. For more information, see the Dakota User's and Theory Manuals.

By specifying this command, each discrepancy function will take the form of a polynomial regression. They will all have the same order.

7.2.68.12.1.4 basis_order

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [discrepancy_type](#)
- [polynomial](#)
- [basis_order](#)

Basis function order of the model discrepancy

Specification

Alias: none

Argument(s): INTEGER

Default: 2

Description

The order of the polynomial for the model discrepancy may be specified using the `basis_order` command, followed by either 0, for which a constant polynomial will be used, 1, for which a linear polynomial will be used, or 2, for which a quadratic polynomial will be used.

Default Behavior

By default, a quadratic polynomial is used.

7.2.68.12.2 `num_prediction_configs`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [num_prediction_configs](#)

Specify number of prediction locations for model discrepancy

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

Following the calculation of the model discrepancy functions, δ_i , and the corrected model $M_i + \delta_i$ may be calculated at new or "prediction" configurations. Using the `num_prediction_configurations` command, the user may specify the number of prediction configurations they wish to be computed. The new configuration locations will be spread uniformly in the configuration variable domain specified in the `variables` block of the input file.

Usage Tips

If none of `num_prediction_configs`, `prediction_configs`, or `import_prediction_configs` is specified, the number of prediction configurations is set to 10 for scalar responses. For field responses, the corrected model will be output at the input values of the configuration variables.

7.2.68.12.3 `prediction_configs`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)

- [prediction_configs](#)

List prediction locations for model discrepancy

Specification

Alias: none

Argument(s): REALLIST

Description

Following the calculation of the model discrepancy functions, δ_i , and the corrected model $M_i + \delta_i$ may be calculated at new or "prediction" configurations. Using the `prediction_configs` command, the user may explicitly list the locations of the prediction configurations they wish to be computed. See parent command for an example.

Usage Tips

If none of `num_prediction_configs`, `prediction_configs`, or `import_prediction_configs` is specified, the number of prediction configurations is set to 10 for scalar responses. For field responses, the corrected model will be output at the input values of the configuration variables.

7.2.68.12.4 import_prediction_configs

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [import_prediction_configs](#)

Specify text file containing prediction configurations for model discrepancy

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
--	--	--	--------------------------	------------------------------

Description

Enables text file import of user-specified prediction configurations at which to calculate the corrected model $M_i + \delta_i$

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_prediction_configs` file must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.68.12.4.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [import_prediction_configs](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.68.12.4.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)

- [import_prediction_configs](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.4.3 `eval_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [import_prediction_configs](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.4.4 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [import_prediction_configs](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.4.5 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [import_prediction_configs](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.12.4.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [import_prediction_configs](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

          0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.12.5 export_discrepancy_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_discrepancy_file](#)

Output file for prediction discrepancy calculations

Specification

Alias: none

Argument(s): STRING

Default: discrepancy export to default filename

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Enables the user to specify the name and format of the file to which prediction discrepancy information is output. The file will contain the values of the discrepancy model δ_i calculated at those points specified with `num_predictions`, `prediction_configs`, or `import_prediction_configs`

Default Behavior

If `export_discrepancy_file` is not specified, information will be automatically output to `dakota_discrepancy_tabular.dat`.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.68.12.5.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_discrepancy_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>eval_id</code>	Enable evaluation ID column in custom-annotated tabular file
	Optional		<code>interface_id</code>	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.68.12.5.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_discrepancy_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.5.3 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_discrepancy_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.5.4 interface_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_discrepancy_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent [custom_annotated](#)

7.2.68.12.5.5 annotated

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_discrepancy_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.12.5.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_discrepancy_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857  0.2601620081  0.759955
0.89991          1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.12.6 export_corrected_model_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_model_file](#)

Output file for corrected model prediction calculations

Specification

Alias: none

Argument(s): STRING

Default: corrected model export to default filename

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Enables the user to specify the name and format of the file to which corrected model $M_i + \delta_i$ information is output. The file will contain the values of the discrepancy model δ_i calculated at those points specified with `num_predictions`, `prediction_configs`, or `import_prediction_configs`.

Corresponding variance information is also output to file, see [export_corrected_variance_file](#)

Default Behavior

If `export_corrected_model_file` is not specified, information will be automatically output to `dakota_corrected_model_tabular.dat`.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.68.12.6.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_model_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.12.6.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_model_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.6.3 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_model_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.6.4 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_model_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.6.5 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_model_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.12.6.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_model_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.12.7 export_corrected_variance_file

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_variance_file](#)

Output file for prediction variance calculations

Specification

Alias: none

Argument(s): STRING

Default: corrected model variance export to default filename

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Enables the user to specify the name and format of the file to which corrected model variance information is output. The file will contain the values of the variances computed at those points specified with `num_predictions`, `prediction_configs`, or `import_prediction_configs`.

For scalar responses, the prediction variance for each observable d_i is calculated according to

$$\Sigma_{total}(x) = \Sigma_{\delta}(x) + \sigma_{exp}^2 I.$$

Here, $\Sigma_{\delta}(x)$ is the (co)variance calculated from the Gaussian process or polynomial regression model, details of which can be found in the Dakota User's and Theory Manuals. If experimental variance information is provided for the calibration, the maximum variance found for the observable d_i is used as σ_{exp}^2 . For more details, see the Dakota User's and Theory Manuals. For field responses, the variance of the discrepancy model alone is output to this file.

Default Behavior

If `export_corrected_variance_file` is not specified, information will be automatically output to `dakota_discrepancy_variance_tabular.dat`.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.68.12.7.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_variance_file](#)

- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and eval_id (no interface_id), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081   0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081   0.760045
...
```

7.2.68.12.7.2 header

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_variance_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.7.3 eval_id

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_variance_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.7.4 `interface_id`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_variance_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.68.12.7.5 `annotated`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_variance_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.68.12.7.6 freeform

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_discrepancy](#)
- [export_corrected_variance_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857  0.2601620081  0.759955
0.89991         1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.68.13 sub_sampling_period

- [Keywords Area](#)

- [method](#)
- [bayes_calibration](#)
- [sub_sampling_period](#)

Specify a sub-sampling of the MCMC chain

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

If a `sub_sampling_period` is specified, the MCMC chain is filtered such that only the sample at the beginning of each period is in the final MCMC chain. The `sub_sampling_period` should therefore be greater than or equal to the correlation length of the samples.

Default Behavior

If not specified, all MCMC samples are included in the final chain.

Expected Output

If specified, an additional tabular output file containing the final (sub- sampled) MCMC chain is generated.

Examples

Generate and then filter a 1000 sample MCMC chain, retaining the first accepted sample and every 100th sample thereafter.

An example method block of a Dakota input file is given below

```
method,  
  bayes_calibration queso  
    chain_samples = 1000 seed = 348  
    dram  
  proposal_covariance  
    diagonal values 1.0e6 1.0e-1  
  sub_sampling_period 100
```

7.2.68.14 probability_levels

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [probability_levels](#)

Specify probability levels at which to compute credible and prediction intervals

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	This capability is currently inactive

Description

Credible and prediction intervals of model responses are computed for specified probabilities. Credible intervals are calculated from the response function values corresponding to the final MCMC chain. Calculation of prediction intervals consider these response values as well as the experimental uncertainty, which is specified by the user via the `experiment_variance_type` command.

Expected Output

If `probability_levels` is specified, Dakota will create a table containing the credible intervals for each response function. The corresponding table containing the prediction intervals will also be created if a `experiment_variance_type` has been specified. This information is output to the screen and to a file. In addition, the output file contains the means and standard deviations of each response function and Gaussian approximations of the 5/95 credible and prediction intervals, in which the lower bound is two standard deviations below the mean and the upper bound is two standard deviations above the mean.

Usage Tips

Only one probability level needs to be specified for each desired interval. Both corresponding end points of the intervals are automatically calculated. For example, if 0.05 is specified, both the 0.05 and 0.95 probability levels are output to the screen and output file.

Additional Discussion

Credible intervals propagate uncertainties in parameter density information to the quantity of interest and quantify how well the model fits the provided data. Prediction intervals propagate both parameter and experimental measurement uncertainties and contain the next experimental or simulated observation with the specified probability.

Examples

Below is a Dakota input file specifying the calculation of credible and prediction intervals

```
method,
  bayes_calibration queso
  chain_samples = 1000 seed = 348
  dram
  proposal_covariance
  diagonal values 1.0e6 1.0e-1
  probability_levels 0.05 0.1
  0.075 0.1

variables,
  uniform_uncertain 2
  upper_bounds 1.e8 10.0
  lower_bounds 1.e6 0.1
  initial_point 2.85e7 2.5
  descriptors 'E' 'w'
  continuous_state 4
  initial_state 3 40000 500 1000
  descriptors 't' 'R' 'X' 'Y'
```

```

interface,
  direct
  analysis_driver = 'mod_cantilever'

responses,
  calibration_terms = 2
  calibration_data_file = 'dakota_cantilever_queso.withsigma.dat'
  freeform
  num_experiments = 10
  experiment_variance_type = 'scalar'
  descriptors = 'stress' 'displacement'
  no_gradients
  no_hessians

```

The resulting screen output below shows the table of credible and prediction intervals.

```

Credibility Intervals for stress
  Response Level      Probability Level
-----
  2.4764049695e+03   5.0000000000e-02
  2.8242874802e+03   9.5000000000e-01
  2.4990608791e+03   1.0000000000e-01
  2.7952985803e+03   9.0000000000e-01

Credibility Intervals for displacement
  Response Level      Probability Level
-----
  2.7409870925e-01   7.5000000000e-02
  3.0991296255e-01   9.2500000000e-01
  2.7538816802e-01   1.0000000000e-01
  3.0889319332e-01   9.0000000000e-01

Prediction Intervals for stress
  Response Level      Probability Level
-----
  2.0964882850e+03   5.0000000000e-02
  3.1993026765e+03   9.5000000000e-01
  2.1822183238e+03   1.0000000000e-01
  3.1099058450e+03   9.0000000000e-01

Prediction Intervals for displacement
  Response Level      Probability Level
-----
  2.3559036055e-01   7.5000000000e-02
  3.5097481218e-01   9.2500000000e-01
  2.4016170870e-01   1.0000000000e-01
  3.4701712866e-01   9.0000000000e-01

```

7.2.68.14.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [probability_levels](#)
- [num_probability_levels](#)

This capability is currently inactive

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

7.2.68.15 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.68.16 `max_iterations`

- [Keywords Area](#)
- [method](#)

- [bayes_calibration](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.68.17 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```
analysis_driver = 'text_book'  
  
responses  
  response_functions = 3  
  no_gradients  
  no_hessians
```

7.2.68.18 scaling

- [Keywords Area](#)
- [method](#)
- [bayes_calibration](#)
- [scaling](#)

Turn on scaling for variables, responses, and constraints

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): none

Default: no scaling

Description

Some optimization and calibration methods support scaling of continuous design variables, objective functions, calibration terms, and constraints. This is activated with the `scaling` keyword. Discrete variable scaling is not supported.

When scaling is enabled, variables, functions, gradients, Hessians, etc., are transformed such that the method iterates in scaled variable space, whereas evaluations of the computational model as specified in the interface are performed on the original problem scale. Therefore using scaling does not require rewriting the interface to the simulation code.

Scaling also requires the specification of additional keywords `scale_types` and/or `*scales` in the [variables](#) and [responses](#) blocks. When the `scaling` keyword is omitted from [method](#), these scaling type and value specifications are ignored in the variables and responses sections.

This page describes the usage of all scaling related keywords. The additional keywords come in pairs, one pair for each set of quantities (variables, primary responses, or constraints) to be scaled.

- a `*scales` keyword, which gives characteristic values (divisors)
- a `*scale_types` keyword, which determines how to use the characteristic values

The pair of keywords both take argument(s), and the length of the arguments can be either be one or equal to the number of quantities to be scaled (see details in [responses](#) for lengths when field responses are active). If one argument is given, it will apply to all quantities in the set. See the examples below.

Scale Types

There are three scale types:

1. `value` - characteristic value scaling

The target quantity will be scaled (divided) by the specified characteristic value.

2. `auto` - automatic scaling

First the quantity is scaled by any characteristic value, then automatic scaling will be attempted according to the following scheme:

- two-sided bounds scaled into the interval [0,1];
- one-sided bound or targets are scaled by the characteristic value, moving the bound or target to 1 and changing the sense of inequalities where necessary;
- no bounds or targets: no automatic scaling possible, therefore no scaling for this component

Automatic scaling is not available for objective functions nor calibration terms since they lack bound constraints. Further, when automatically scaled, linear constraints are scaled by characteristic values only, not affinely scaled into [0,1].

3. `log` - logarithmic scaling

First, any characteristic values from the optional `*scales` specification are applied. Then logarithm base 10 scaling is applied.

Logarithmic scaling is not available for linear constraints.

When continuous design variables are log scaled, linear constraints are not allowed.

Scales

The `*scales` keywords are used to specify the characteristic values. These must be non-zero real numbers. The numbers are used according to the corresponding `*scale_types`, as described above. A value of 1.0 can be used to selectively omit some quantities in a vector from being scaled.

Depending on the scale type, the characteristic values may be required or optional.

- `auto`, `log` - optional
- `value` - required.

A warning is issued if scaling would result in division by a value smaller in magnitude than $1.0e10 * \text{DBL_MIN}$. User-provided values violating this lower bound are accepted unaltered, whereas for automatically calculated scaling, the lower bound is enforced.

Examples

The two examples below are equivalent:

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value"
primary_scales = 1 1 100
```

```
responses
objective_functions 3
sense "maximize"
primary_scale_types = "value" "value" "value"
primary_scales = 1 1 100
```

7.2.69 dace

- [Keywords Area](#)
- [method](#)
- [dace](#)

Topics

This keyword is related to the topics:

- [package_ddace](#)
- [design_and_analysis_of_computer_experiments](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
			grid	Grid Sampling
	Required (<i>Choose One</i>)	DACE type (Group 1)	random	Uses purely random Monte Carlo sampling to sample variables
			oas	Orthogonal Array Sampling
			lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			oa_lhs	Orthogonal Array Latin Hypercube Sampling
			box_behnken	Box-Behnken Design
			central_composite	Central Composite Design
	Optional		samples	Number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets

	Optional	main_effects	ANOVA
	Optional	quality_metrics	Calculate metrics to assess the quality of quasi-Monte Carlo samples
	Optional	variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into contributions from variables
	Optional	symbols	Number of replications in the sample set
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The Distributed Design and Analysis of Computer Experiments (DDACE) library provides the following DACE techniques:

1. grid sampling ([grid](#))
2. pure random sampling ([random](#))
3. orthogonal array sampling ([oas](#))
4. latin hypercube sampling ([lhs](#))
5. orthogonal array latin hypercube sampling ([oa_lhs](#))
6. Box-Behnken ([box_behnken](#))
7. central composite design ([central_composite](#))

These methods all generate point sets that may be used to drive a set of computer experiments. Note that all of the DACE methods generated randomized designs, except for Box-Behnken and Central composite which are classical designs. That is, the grid sampling will generate a randomized grid, not what one typically thinks of as a grid of uniformly spaced points over a rectangular grid. Similar, the orthogonal array is a randomized version of an orthogonal array: it does not generate discrete, fixed levels.

In addition to the selection of the method, there are keywords that affect the method outputs:

1. [main_effects](#)
2. [quality_metrics](#)
3. [variance_based_decomp](#)

And keywords that affect the sampling:

1. [fixed_seed](#)
2. [symbols](#)
3. [samples](#)
4. [seed](#)

See Also

These keywords may also be of interest:

- [fsu_cvt](#)
- [fsu_quasi_mc](#)
- [psuade_moat](#)

7.2.69.1 grid

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [grid](#)

Grid Sampling

Specification

Alias: none

Argument(s): none

Description

The grid option in DACE will produce a randomized grid of points. If you are interested in a regular grid of points, use the multidimensional parameter study (under Parameter Studies) instead. Grid Sampling

7.2.69.2 random

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.69.3 oas

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [oas](#)

Orthogonal Array Sampling

Specification

Alias: none

Argument(s): none

Description

Orthogonal array sampling (OAS) is a widely used technique for running experiments and systematically testing factor effects. An orthogonal array sample can be described as a 4-tuple $(m;n;s;r)$, where m is the number of sample points, n is the number of input variables, s is the number of symbols, and r is the strength of the orthogonal array. The number of sample points, m , must be a multiple of the number of symbols, s . The number of symbols refers to the number of levels per input variable. The strength refers to the number of columns where we are guaranteed to see all the possibilities an equal number of times. Note that the DACE OAS capability produces a randomized orthogonal array: the samples for a particular level are randomized within that level.

If one examines the sample sets in an orthogonal array by looking at the rows as individual samples and columns as the variables sampled, one sees that the columns are orthogonal to each other in an orthogonal array. This feature is important in main effects analysis, which is a sensitivity analysis technique that identifies which variables have the most influence on the output.

7.2.69.4 lhs

- [Keywords Area](#)

- [method](#)
- [dace](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.69.5 `oa_lhs`

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [oa_lhs](#)

Orthogonal Array Latin Hypercube Sampling

Specification

Alias: none

Argument(s): none

Description

The Orthogonal Array Latin Hypercube Sampling option in DACE produces a "latinized" version of an orthogonal array. That is, after the orthogonal array is generated, the samples go through a stratification process to produce samples that have been both orthogonalized and stratified.

See Also

These keywords may also be of interest:

- [oas](#)

7.2.69.6 box_behnken

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [box_behnken](#)

Box-Behnken Design

Specification

Alias: none

Argument(s): none

Description

The Box-Behnken design is similar to a Central Composite design, with some differences. The Box-Behnken design is a quadratic design in that it does not contain an embedded factorial or fractional factorial design. In this design the treatment combinations are at the midpoints of edges of the process space and at the center, as compared with CCD designs where the extra points are placed at 'star points' on a circle outside of the process space. Box-Behnken designs are rotatable (or near rotatable) and require 3 levels of each factor.

7.2.69.7 central_composite

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [central_composite](#)

Central Composite Design

Specification

Alias: none

Argument(s): none

Description

A central composite design (CCD), contains an embedded factorial or fractional factorial design with a center points that is augmented with a group of "star points" that allow estimation of curvature.

Examples

See the User's Manual for an example.

7.2.69.8 samples

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10 \cdot \text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N \cdot (\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.69.9 seed

- [Keywords Area](#)
- [method](#)

- [dace](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.69.10 `fixed_seed`

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.69.11 main_effects

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [main_effects](#)

ANOVA

Specification

Alias: none

Argument(s): none

Default: No main_effects

Description

The `main_effects` control prints Analysis-of-Variance main effects results (e.g. ANOVA tables with p-values per variable). The `main_effects` control is only operational with the orthogonal arrays or Latin Hypercube designs, not for Box Behnken or Central Composite designs.

Main effects is a sensitivity analysis method which identifies the input variables that have the most influence on the output. In main effects, the idea is to look at the mean of the response function when variable A (for example) is at level 1 vs. when variable A is at level 2 or level 3. If these mean responses of the output are statistically significantly different at different levels of variable A, this is an indication that variable A has a significant effect on the response. The orthogonality of the columns is critical in performing main effects analysis, since the column orthogonality means that the effects of the other variables "cancel out" when looking at the overall effect from one variable at its different levels.

7.2.69.12 quality_metrics

- [Keywords Area](#)

- [method](#)
- [dace](#)
- [quality_metrics](#)

Calculate metrics to assess the quality of quasi-Monte Carlo samples

Topics

This keyword is related to the topics:

- [package_fsudace](#)

Specification

Alias: none

Argument(s): none

Default: No quality_metrics

Description

`quality_metrics` calculates four quality metrics relating to the volumetric spacing of the samples. The four quality metrics measure different aspects relating to the uniformity of point samples in hypercubes. Desirable properties of such point samples are:

- are the points equally spaced
- do the points cover the region
- and are they isotropically distributed
- with no directional bias in the spacing

The four quality metrics we report are:

- `h`: the point distribution norm, which is a measure of uniformity of the point distribution
- `chi`: a regularity measure, and provides a measure of local uniformity of a set of points
- `tau`: the second moment trace measure
- `d`: the second moment determinant measure

All of these values are scaled so that smaller is better (the smaller the metric, the better the uniformity of the point distribution).

Examples

Complete explanation of these measures can be found in [39].

7.2.69.13 `variance_based_decomp`

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into contributions from variables

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance based decomposition using the keyword `variance_based_decomp`. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

Default Behavior

Because of the computational cost, `variance_based_decomp` is turned off as a default.

If the user specified a number of samples, N , and a number of nondeterministic variables, M , variance-based decomposition requires the evaluation of $N*(M+2)$ samples. **Note that specifying this keyword will increase the number of function evaluations above the number requested with the `samples` keyword since replicated sets of sample values are evaluated.**

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response

Usage Tips

To obtain sensitivity indices that are reasonably accurate, we recommend that N , the number of samples, be at least one hundred and preferably several hundred or thousands.

Examples

```
method,
  sampling
  sample_type lhs
```

```
samples = 100
variance_based_decomp
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [76] and [90].

7.2.69.13.1 drop_tolerance

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

7.2.69.14 `symbols`

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [symbols](#)

Number of replications in the sample set

Specification

Alias: none

Argument(s): INTEGER

Default: default for sampling algorithm

Description

`symbols` is related to the number of levels per variable in the sample set (a larger number of symbols equates to more stratification and fewer replications). For example, if `symbols = 7`, each variable would be divided into seven levels.

7.2.69.15 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [dace](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.70 fsu_cvt

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)

Design of Computer Experiments - Centroidal Voronoi Tessellation

Topics

This keyword is related to the topics:

- [package_fsudace](#)
- [design_and_analysis_of_computer_experiments](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		samples	Number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional		fixed_seed	Reuses the same seed value for multiple random sampling sets
	Optional		latinize	Adjust samples to improve the discrepancy of the marginal distributions
	Optional		quality_metrics	Calculate metrics to assess the quality of quasi-Monte Carlo samples

	Optional	variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into contributions from variables
	Optional	trial_type	Specify how the trial samples are generated
	Optional	num_trials	The number of secondary sample points generated to adjust the location of the primary sample points
	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	model_pointer	Identifier for model block to be used by a method

Description

The FSU Centroidal Voronoi Tessellation method (`fsu_cvt`) produces a set of sample points that are (approximately) a Centroidal Voronoi Tessellation. The primary feature of such a set of points is that they have good volumetric spacing; the points tend to arrange themselves in a pattern of cells that are roughly the same shape.

To produce this set of points, an almost arbitrary set of initial points is chosen, and then an internal set of iterations is carried out. These iterations repeatedly replace the current set of sample points by an estimate of the centroids of the corresponding Voronoi subregions. [18].

The user may generally ignore the details of this internal iteration. If control is desired, however, there are a few variables with which the user can influence the iteration. The user may specify:

- [max_iterations](#), the number of iterations carried out
- [num_trials](#), the number of secondary sample points generated to adjust the location of the primary sample points
- [trial_type](#), which controls how these secondary sample points are generated

This method generates sets of uniform random variables on the interval [0,1]. If the user specifies lower and upper bounds for a variable, the [0,1] samples are mapped to the [lower, upper] interval.

Theory

This method is designed to generate samples with the goal of low discrepancy. Discrepancy refers to the nonuniformity of the sample points within the hypercube.

Discrepancy is defined as the difference between the actual number and the expected number of points one would expect in a particular set B (such as a hyper-rectangle within the unit hypercube), maximized over all such sets. Low discrepancy sequences tend to cover the unit hypercube reasonably uniformly.

Centroidal Voronoi Tessellation does very well volumetrically: it spaces the points fairly equally throughout the space, so that the points cover the region and are isotropically distributed with no directional bias in the point placement. There are various measures of volumetric uniformity which take into account the distances between pairs of points, regularity measures, etc. Note that Centroidal Voronoi Tessellation does not produce low-discrepancy sequences in lower dimensions. The lower-dimension (such as 1-D) projections of Centroidal Voronoi Tessellation can have high discrepancy.

See Also

These keywords may also be of interest:

- [dace](#)
- [fsu_quasi_mc](#)
- [psuade_moat](#)

7.2.70.1 samples

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10*\text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N*(\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```


7.2.70.2 seed

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.70.3 fixed_seed

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [fixed_seed](#)

Reuses the same seed value for multiple random sampling sets

Specification

Alias: none

Argument(s): none

Default: not fixed; pattern varies run-to-run

Description

The `fixed_seed` flag is relevant if multiple sampling sets will be generated over the course of a Dakota analysis. This occurs when using advance methods (e.g., surrogate-based optimization, optimization under uncertainty). The same seed value is reused for each of these multiple sampling sets, which can be important for reducing variability in the sampling results.

Default Behavior

The default behavior is to not use a fixed seed, as the repetition of the same sampling pattern can result in a modeling weakness that an optimizer could potentially exploit (resulting in actual reliabilities that are lower than the estimated reliabilities). For repeatable studies, the `seed` must also be specified.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    fixed_seed
```

7.2.70.4 latinize

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [latinize](#)

Adjust samples to improve the discrepancy of the marginal distributions

Specification

Alias: none

Argument(s): none

Default: No latinization

Description

The `latinize` control takes the samples and "latinizes" them, meaning that each original sample is moved so that it falls into one strata or bin in each dimension as in Latin Hypercube sampling. The default setting is NOT to latinize. However, one may be interested in doing this in situations where one wants better discrepancy of the 1-dimensional projections (the marginal distributions).

7.2.70.5 `quality_metrics`

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [quality_metrics](#)

Calculate metrics to assess the quality of quasi-Monte Carlo samples

Topics

This keyword is related to the topics:

- [package_fsudace](#)

Specification

Alias: none

Argument(s): none

Default: No `quality_metrics`

Description

`quality_metrics` calculates four quality metrics relating to the volumetric spacing of the samples. The four quality metrics measure different aspects relating to the uniformity of point samples in hypercubes. Desirable properties of such point samples are:

- are the points equally spaced
- do the points cover the region
- and are they isotropically distributed
- with no directional bias in the spacing

The four quality metrics we report are:

- `h`: the point distribution norm, which is a measure of uniformity of the point distribution
- `chi`: a regularity measure, and provides a measure of local uniformity of a set of points
- `tau`: the second moment trace measure
- `d`: the second moment determinant measure

All of these values are scaled so that smaller is better (the smaller the metric, the better the uniformity of the point distribution).

Examples

Complete explanation of these measures can be found in [39].

7.2.70.6 `variance_based_decomp`

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into contributions from variables

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance based decomposition using the keyword `variance_based_decomp`. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

Default Behavior

Because of the computational cost, `variance_based_decomp` is turned off as a default.

If the user specified a number of samples, N, and a number of nondeterministic variables, M, variance-based decomposition requires the evaluation of $N*(M+2)$ samples. **Note that specifying this keyword will increase the number of function evaluations above the number requested with the `samples` keyword since replicated sets of sample values are evaluated.**

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response

Usage Tips

To obtain sensitivity indices that are reasonably accurate, we recommend that N, the number of samples, be at least one hundred and preferably several hundred or thousands.

Examples

```
method,
  sampling
  sample_type lhs
```

```
samples = 100
variance_based_decomp
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76]: "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [76] and [90].

7.2.70.6.1 drop_tolerance

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
  sample_type lhs
  samples = 100
  variance_based_decomp
  drop_tolerance = 0.001
```

7.2.70.7 trial_type

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [trial_type](#)

Specify how the trial samples are generated

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Trial Type (Group 1)	grid	Samples on a regular grid
			halton	Generate samples from a Halton sequence
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The user has the option to specify the method by which the trials are created to adjust the centroids. The `trial_type` can be one of three types:

- `random`, where points are generated randomly
- `halton`, where points are generated according to the Halton sequence
- `grid`, where points are placed on a regular grid over the hyperspace.

7.2.70.7.1 grid

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [trial_type](#)
- [grid](#)

Samples on a regular grid

Specification

Alias: none

Argument(s): none

Description

Points are placed on a regular grid over the hyperspace.

See Also

These keywords may also be of interest:

- [trial_type](#)

7.2.70.7.2 halton

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [trial_type](#)
- [halton](#)

Generate samples from a Halton sequence

Topics

This keyword is related to the topics:

- [package_fsudace](#)

Specification

Alias: none

Argument(s): none

Description

The quasi-Monte Carlo sequences of Halton are deterministic sequences determined by a set of prime bases. These sequences generate random numbers with the goal of filling a unit hypercube uniformly.

Generally, we recommend that the user leave the default setting for the bases, which are the lowest primes. Thus, if one wants to generate a sample set for 3 random variables, the default bases used are 2, 3, and 5 in the Halton sequence. To give an example of how these sequences look, the Halton sequence in base 2 starts with points 0.5, 0.25, 0.75, 0.125, 0.625, etc. The first few points in a Halton base 3 sequence are 0.33333, 0.66667, 0.11111, 0.44444, 0.77777, etc. Notice that the Halton sequence tends to alternate back and forth, generating a point closer to zero then a point closer to one. An individual sequence is based on a radix inverse function defined on a prime base. The prime base determines how quickly the [0,1] interval is filled in.

Theory

For more information about these sequences, see [44], [45], and [1].

7.2.70.7.3 random

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [trial_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.2.70.8 num_trials

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [num_trials](#)

The number of secondary sample points generated to adjust the location of the primary sample points

Specification

Alias: none

Argument(s): INTEGER

Default: 10000

Description

In general, the variable with the most influence on the quality of the final sample set is `num_trials`, which determines how well the Voronoi subregions are sampled.

Generally, `num_trials` should be "large", certainly much bigger than the number of sample points being requested; a reasonable value might be 10,000, but values of 100,000 or 1 million are not unusual.

7.2.70.9 `max_iterations`

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: $25*n$)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.70.10 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [fsu_cvt](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```
surrogate global,  
dace_method_pointer = 'DACE'  
polynomial quadratic  
  
method  
id_method = 'DACE'  
model_pointer = 'DACE_M'  
sampling sample_type lhs  
samples = 121 seed = 5034 rng rnum2  
  
model  
id_model = 'DACE_M'  
single  
interface_pointer = 'I1'  
  
variables  
uniform_uncertain = 2  
lower_bounds = 0. 0.  
upper_bounds = 1. 1.  
descriptors = 'x1' 'x2'  
  
interface  
id_interface = 'I1'  
system async evaluation_concurrency = 5  
analysis_driver = 'text_book'  
  
responses  
response_functions = 3  
no_gradients  
no_hessians
```

7.2.71 psuade_moat

- [Keywords Area](#)
- [method](#)
- [psuade_moat](#)

Morris One-at-a-Time

Topics

This keyword is related to the topics:

- [package_psuade](#)
- [design_and_analysis_of_computer_experiments](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		partitions	Number of partitions of each variable
	Optional		samples	Number of samples for sampling-based methods
	Optional		seed	Seed of the random number generator
	Optional		model_pointer	Identifier for model block to be used by a method

Description

The Morris One-At-A-Time (MOAT) method, originally proposed by Morris [64], is a screening method, designed to explore a computational model to distinguish between input variables that have negligible, linear and additive, or nonlinear or interaction effects on the output. The computer experiments performed consist of individually randomized designs which vary one input factor at a time to create a sample of its elementary effects.

The number of samples (`samples`) must be a positive integer multiple of (number of continuous design variable + 1) and will be automatically adjusted if misspecified.

The number of partitions (`partitions`) applies to each variable being studied and must be odd (the number of MOAT levels per variable is `partitions + 1`). This will also be adjusted at runtime as necessary.

For information on practical use of this method, see [76].

Theory

With MOAT, each dimension of a k -dimensional input space is uniformly partitioned into p levels, creating a grid of p^k points $\mathbf{x} \in \mathbf{R}^k$ at which evaluations of the model $y(\mathbf{x})$ might take place. An elementary effect corresponding to input i is computed by a forward difference

$$d_i(\mathbf{x}) = \frac{y(\mathbf{x} + \Delta \mathbf{e}_i) - y(\mathbf{x})}{\Delta},$$

where \mathbf{e}_i is the i^{th} coordinate vector, and the step Δ is typically taken to be large (this is not intended to be a local derivative approximation). In the present implementation of MOAT, for an input variable scaled to $[0, 1]$, $\Delta = \frac{p}{2(p-1)}$, so the step used to find elementary effects is slightly larger than half the input range.

The distribution of elementary effects d_i over the input space characterizes the effect of input i on the output of interest. After generating r samples from this distribution, their mean,

$$\mu_i = \frac{1}{r} \sum_{j=1}^r d_i^{(j)}$$

modified mean

$$\mu_i^* = \frac{1}{r} \sum_{j=1}^r |d_i^{(j)}|,$$

(using absolute value) and standard deviation

$$\sigma_i = \sqrt{\frac{1}{r} \sum_{j=1}^r (d_i^{(j)} - \mu_i)^2}$$

are computed for each input i . The mean and modified mean give an indication of the overall effect of an input on the output. Standard deviation indicates nonlinear effects or interactions, since it is an indicator of elementary effects varying throughout the input space.

7.2.71.1 partitions

- [Keywords Area](#)
- [method](#)
- [psuade_moat](#)
- [partitions](#)

Number of partitions of each variable

Specification

Alias: none

Argument(s): INTEGERLIST

Default: 3

Description

Described on the parent page, [psuade_moat](#)

7.2.71.2 samples

- [Keywords Area](#)
- [method](#)
- [psuade_moat](#)
- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10 \cdot \text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N \cdot (\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.71.3 seed

- [Keywords Area](#)
- [method](#)
- [psuade_moat](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.71.4 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [psuade_moat](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```
surrogate global,  
dace_method_pointer = 'DACE'  
polynomial quadratic  
  
method  
id_method = 'DACE'  
model_pointer = 'DACE_M'  
sampling sample_type lhs  
samples = 121 seed = 5034 rng rnum2  
  
model  
id_model = 'DACE_M'  
single  
interface_pointer = 'I1'  
  
variables  
uniform_uncertain = 2  
lower_bounds = 0. 0.  
upper_bounds = 1. 1.  
descriptors = 'x1' 'x2'  
  
interface  
id_interface = 'I1'  
system async evaluation_concurrency = 5  
analysis_driver = 'text_book'  
  
responses  
response_functions = 3  
no_gradients  
no_hessians
```

7.2.72 local_evidence

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)

Evidence theory with evidence measures computed with local optimization methods

Topics

This keyword is related to the topics:

- [epistemic_uncertainty_quantification_methods](#)
- [evidence_theory](#)

Specification

Alias: nond_local_evidence

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Optimization Solver (Group 1)	sqp	Uses a sequential quadratic programming method for underlying optimization
			nip	Uses a nonlinear interior point method for underlying optimization
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		model_pointer	Identifier for model block to be used by a method

Description

Two local optimization methods are available: [sqp](#) (sequential quadratic programming) or [nip](#) (nonlinear interior point method).

Additional Resources

See the topic page [evidence_theory](#) for important background information and usage notes.

Refer to [variable_support](#) for information on supported variable types.

See Also

These keywords may also be of interest:

- [global_evidence](#)
- [global_interval_est](#)

- [local_interval_est](#)

7.2.72.1 sqp

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [sqp](#)

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `sqp` keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

7.2.72.2 nip

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [nip](#)

Uses a nonlinear interior point method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

7.2.72.3 response_levels

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
-----	-----	-----
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12

```

3.6000000000e+11 4.0000000000e+11 4.2500000000e-12
4.0000000000e+11 4.4000000000e+11 3.7500000000e-12
4.4000000000e+11 5.4196114379e+11 2.2557612778e-12
PDF for response_fn_2:
  Bin Lower          Bin Upper          Density Value
  -----
4.6431154744e+04 6.0000000000e+04 2.8742313192e-05
6.0000000000e+04 6.5000000000e+04 6.4000000000e-05
6.5000000000e+04 7.0000000000e+04 4.0000000000e-05
7.0000000000e+04 7.8702465755e+04 1.0341896485e-05
PDF for response_fn_3:
  Bin Lower          Bin Upper          Density Value
  -----
2.3796737090e+05 3.5000000000e+05 4.2844660868e-06
3.5000000000e+05 4.0000000000e+05 8.6000000000e-06
4.0000000000e+05 4.5000000000e+05 1.8000000000e-06

```

Level mappings for each response function:

```

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
3.6000000000e+11 5.5000000000e-01
4.0000000000e+11 3.8000000000e-01
4.4000000000e+11 2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
6.0000000000e+04 6.1000000000e-01
6.5000000000e+04 2.9000000000e-01
7.0000000000e+04 9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
  -----
3.5000000000e+05 5.2000000000e-01
4.0000000000e+05 9.0000000000e-02
4.5000000000e+05 0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.72.3.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.72.3.2 compute

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels

			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.72.3.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.72.3.2.2 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

7.2.72.3.2.3 system

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.72.3.2.4 series

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.72.3.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.72.4 probability_levels

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	Specify which probability_levels correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations  = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
```

```

uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors        = 'TF1u' 'TF2u'
weibull_uncertain = 2
  alphas            = 12., 30.
  betas              = 250., 590.
  descriptors        = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
  num_pairs         = 3      4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors        = 'TF1h' 'TF2h'
histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors        = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		

```
3.6997214153e+05 3.0000000000e-01
3.8100966235e+05 2.0000000000e-01
4.4111498127e+05 0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.72.4.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.72.5 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.72.5.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `gen_reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.72.6 distribution

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Distribution Type (CDF/CCDF) (Group 1)	Dakota Keyword cumulative	Dakota Keyword Description Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
```

```
sample_type lhs
samples = 10
distribution cumulative
```

7.2.72.6.1 cumulative

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.72.6.2 complementary

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.72.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [local_evidence](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last `model` block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.73 local_interval_est

- [Keywords Area](#)
- [method](#)
- [local_interval_est](#)

Interval analysis using local optimization

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)
- [epistemic_uncertainty_quantification_methods](#)
- [interval_estimation](#)

Specification

Alias: nond_local_interval_est

Argument(s): none

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Optimization Solver (Group 1)	Dakota Keyword	Dakota Keyword Description
			sqp	Uses a sequential quadratic programming method for underlying optimization
			nip	Uses a nonlinear interior point method for underlying optimization
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

Interval analysis using local methods ([local_interval_est](#)). If the problem is amenable to local optimization methods (e.g. can provide derivatives or use finite difference method to calculate derivatives), then one can use one of two local methods to calculate these bounds.

- [sqp](#)
- [nip](#)

Additional Resources

Refer to [variable_support](#) for information on supported variable types.

Theory

In interval analysis, one assumes that nothing is known about an epistemic uncertain variable except that its value lies somewhere within an interval. In this situation, it is NOT assumed that the value has a uniform probability of occurring within the interval. Instead, the interpretation is that any value within the interval is a possible value or a potential realization of that variable. In interval analysis, the uncertainty quantification problem is one of determining the resulting bounds on the output (defining the output interval) given interval bounds on the inputs. Again, any output response that falls within the output interval is a possible output with no frequency information assigned to it.

See Also

These keywords may also be of interest:

- [global_evidence](#)
- [global_interval_est](#)
- [local_evidence](#)

7.2.73.1 [sqp](#)

- [Keywords Area](#)
- [method](#)
- [local_interval_est](#)
- [sqp](#)

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `sqp` keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

7.2.73.2 nip

- [Keywords Area](#)
- [method](#)
- [local_interval_est](#)
- [nip](#)

Uses a nonlinear interior point method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

7.2.73.3 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [local_interval_est](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DAGE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.73.4 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [local_interval_est](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_` pointer for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
    sample_type lhs
    distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0.  0.
  upper_bounds = 1.  1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.74 local_reliability

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)

Local reliability method

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)
- [reliability_methods](#)

Specification

Alias: nond_local_reliability

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		mpp_search	Specify which MPP search option to use
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional		reliability_levels	Specify reliability levels at which the response values will be estimated
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions

	Optional	max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	final_moments	Output moments of the specified type and include them within the set of final statistics.
	Optional	model_pointer	Identifier for model block to be used by a method

Description

Local reliability methods compute approximate response function distribution statistics based on specified uncertain variable probability distributions. Each of the local reliability methods can compute forward and inverse mappings involving response, probability, reliability, and generalized reliability levels.

The forward reliability analysis algorithm of computing reliabilities/probabilities for specified response levels is called the Reliability Index Approach (RIA), and the inverse reliability analysis algorithm of computing response levels for specified probability levels is called the Performance Measure Approach (PMA).

The different RIA/PMA algorithm options are specified using the `mpp_search` specification which selects among different limit state approximations that can be used to reduce computational expense during the MPP searches.

Theory

The Mean Value method (MV, also known as MVFOSM in [43]) is the simplest, least-expensive method in that it estimates the response means, response standard deviations, and all CDF/CCDF forward/inverse mappings from a single evaluation of response functions and gradients at the uncertain variable means. This approximation can have acceptable accuracy when the response functions are nearly linear and their distributions are approximately Gaussian, but can have poor accuracy in other situations.

All other reliability methods perform an internal nonlinear optimization to compute a most probable point (MPP) of failure. A sign convention and the distance of the MPP from the origin in the transformed standard normal space ("u-space") define the reliability index, as explained in the section on Reliability Methods in the Uncertainty Quantification chapter of the Users Manual [5]. Also refer to [variable_support](#) for additional information on supported variable types for transformations to standard normal space. The reliability can then be converted to a probability using either first- or second-order integration, may then be refined using importance sampling, and finally may be converted to a generalized reliability index.

See Also

These keywords may also be of interest:

- [adaptive_sampling](#)

- [gpais](#)
- [global_reliability](#)
- [sampling](#)
- [importance_sampling](#)
- [polynomial_chaos](#)
- [stoch_collocation](#)

7.2.74.1 mpp_search

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)

Specify which MPP search option to use

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)
- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Default: No MPP search (MV method)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	MPP Approximation (Group 1)	x_taylor_mean	Form Taylor series approximation in "x-space" at variable means
			u_taylor_mean	Form Taylor series approximation in "u-space" at variable means
			x_taylor_mpp	X-space Taylor series approximation with iterative updates

			u_taylor_mpp	U-space Taylor series approximation with iterative updates
			x_two_point	Predict MPP using Two-point Adaptive Nonlinear Approximation in "x-space"
			u_two_point	Predict MPP using Two-point Adaptive Nonlinear Approximation in "u-space"
			x_multi_point	MPP search for local reliability based on QMEA multi-point approximation in u-space
			u_multi_point	MPP search for local reliability based on QMEA multi-point approximation in x-space
			no_approx	Perform MPP search on original response functions (use no approximation)
	Optional (<i>Choose One</i>)	Optimization Solver (Group 2)	sqp	Uses a sequential quadratic programming method for underlying optimization
			nip	Uses a nonlinear interior point method for underlying optimization
	Optional		integration	Integration approach

Description

The `x_taylor_mean` MPP search option performs a single Taylor series approximation in the space of the original uncertain variables ("x-space") centered at the uncertain variable means, searches for the MPP for each response/probability level using this approximation, and performs a validation response evaluation at each predicted MPP. This option is commonly known as the Advanced Mean Value (AMV) method. The `u_taylor_mean` option is identical to the `x_taylor_mean` option, except that the approximation is performed in u-space. The `x_taylor_mpp` approach starts with an x-space Taylor series at the uncertain variable means, but iteratively updates the Taylor series approximation at each MPP prediction until the MPP converges. This option is commonly

known as the AMV+ method. The `u_taylor_mpp` option is identical to the `x_taylor_mpp` option, except that all approximations are performed in u-space. The order of the Taylor-series approximation is determined by the corresponding `responses` specification and may be first or second-order. If second-order (methods named `AMV2` and `AMV2+` in [22]), the series may employ analytic, finite difference, or quasi Hessians (BFGS or SR1). The `x_two_point` MPP search option uses an x-space Taylor series approximation at the uncertain variable means for the initial MPP prediction, then utilizes the Two-point Adaptive Nonlinear Approximation (TANA) outlined in [95] for all subsequent MPP predictions. The `u_two_point` approach is identical to `x_two_point`, but all the approximations are performed in u-space. The `x_taylor_mpp` and `u_taylor_mpp`, `x_two_point` and `u_two_point` approaches utilize the `max_iterations` and `convergence_tolerance` method independent controls to control the convergence of the MPP iterations (the maximum number of MPP iterations per level is limited by `max_iterations`, and the MPP iterations are considered converged when $\| \mathbf{u}^{(k+1)} - \mathbf{u}^{(k)} \|_2 < \text{convergence_tolerance}$). And, finally, the `no_approx` option performs the MPP search on the original response functions without the use of any approximations. The optimization algorithm used to perform these MPP searches can be selected to be either sequential quadratic programming (uses the `npsol_sqp` optimizer) or nonlinear interior point (uses the `optpp_q_newton` optimizer) algorithms using the `sqp` or `nip` keywords.

In addition to the MPP search specifications, one may select among different integration approaches for computing probabilities at the MPP by using the `integration` keyword followed by either `first_order` or `second_order`. Second-order integration employs the formulation of [51] (the approach of [12] and the correction of [52] are also implemented, but are not active). Combining the `no_approx` option of the MPP search with first- and second-order integrations results in the traditional first- and second-order reliability methods (FORM and SORM). These integration approximations may be subsequently refined using importance sampling. The `refinement` specification allows the selection of basic importance sampling (`import`), adaptive importance sampling (`adapt_import`), or multimodal adaptive importance sampling (`mm_adapt_import`), along with the specification of number of samples (`samples`) and random seed (`seed`). Additional details on these methods are available in [24] and [22] and in the Uncertainty Quantification Capabilities chapter of the Users Manual [5].

7.2.74.1.1 `x_taylor_mean`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [x_taylor_mean](#)

Form Taylor series approximation in "x-space" at variable means

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

This `mpp_search` option performs a single Taylor series approximation in the space of the original uncertain variables ("x-space") centered at the uncertain variable means, searches for the MPP for each response/probability level using this approximation, and performs a validation response evaluation at each predicted MPP. This option is commonly known as the Advanced Mean Value (AMV) method.

7.2.74.1.2 `u_taylor_mean`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [u_taylor_mean](#)

Form Taylor series approximation in "u-space" at variable means

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

This `mpp_search` option performs a single Taylor series approximation in the transformed space of the uncertain variables ("u-space") centered at the uncertain variable means. This option is commonly known as the Advanced Mean Value (AMV) method, but is performed in u-space instead of x-space.

7.2.74.1.3 `x_taylor_mpp`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [x_taylor_mpp](#)

X-space Taylor series approximation with iterative updates

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

This `mpp_search` option starts with an x-space Taylor series at the uncertain variable means, but iteratively updates the Taylor series approximation at each MPP prediction until the MPP converges. This option is commonly known as the AMV+ method.

7.2.74.1.4 `u_taylor_mpp`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [u_taylor_mpp](#)

U-space Taylor series approximation with iterative updates

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

This `mpp_search` option starts with a u-space Taylor series at the uncertain variable means, and iteratively updates the Taylor series approximation at each MPP prediction until the MPP converges. This option is commonly known as the AMV+ method and is identify to `x_taylor_mpp` except that it is performed in u-space.

7.2.74.1.5 `x_two_point`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [x_two_point](#)

Predict MPP using Two-point Adaptive Nonlinear Approximation in "x-space"

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

This `mpp_search` option uses an x-space Taylor series approximation at the uncertain variable means for the initial MPP prediction, then utilizes the Two-point Adaptive Nonlinear Approximation (TANA) outlined in [Xu98 "Xu and Grandhi, 1998"] for all subsequent MPP predictions.

7.2.74.1.6 `u_two_point`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [u_two_point](#)

Predict MPP using Two-point Adaptive Nonlinear Approximation in "u-space"

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

This `mpp_search` option is identical to `x_two_point`, but it performs the Two-point Adaptive Nonlinear Approximation (TANA) in u-space instead of x-space.

7.2.74.1.7 `x_multi_point`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [x_multi_point](#)

MPP search for local reliability based on QMEA multi-point approximation in u-space

Specification

Alias: none

Argument(s): none

Description

Local reliability methods that search for the most probable point of failure (MPP) may use a surrogate-based optimization procedure. The `x_multi_point` surrogate option creates a QMEA approximation in the original random variable space ("x-space") from accumulating data from multiple expansion points.

This capability is **experimental**.

7.2.74.1.8 `u_multi_point`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [u_multi_point](#)

MPP search for local reliability based on QMEA multi-point approximation in x-space

Specification

Alias: none

Argument(s): none

Description

Local reliability methods that search for the most probable point of failure (MPP) may use a surrogate-based optimization procedure. The `u_multi_point` surrogate option creates a QMEA approximation in the transformed probability space ("u-space") from accumulating data from multiple expansion points.

This capability is **experimental**.

7.2.74.1.9 `no_approx`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [no_approx](#)

Perform MPP search on original response functions (use no approximation)

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

This `mpp_search` option performs the MPP search on the original response functions without the use of any approximations. Note that the use of the `no_approx` MPP search with first-order probability integration results in the traditional reliability method called FORM (First-Order Reliability Method). Similarly, the use of `no_approx` with second-order probability integration results in SORM (Second-Order Reliability Method).

7.2.74.1.10 `sqp`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [sqp](#)

Uses a sequential quadratic programming method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `sqp` keyword directs Dakota to use a sequential quadratic programming method to solve that problem. A sequential quadratic programming solves a sequence of linearly constrained quadratic optimization problems to arrive at the solution to the optimization problem.

7.2.74.1.11 `nip`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [nip](#)

Uses a nonlinear interior point method for underlying optimization

Specification

Alias: none

Argument(s): none

Description

Many uncertainty quantification methods solve a constrained optimization problem under the hood. The `nip` keyword directs Dakota to use a nonlinear interior point to solve that problem. A nonlinear interior point method traverses the interior of the feasible region to arrive at the solution to the optimization problem.

7.2.74.1.12 integration

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)

Integration approach

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Default: First-order integration

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Integration Order (Group 1)	first_order	First-order integration scheme
			second_order	Second-order integration scheme
	Optional		probability_refinement	Allow refinement of probability and generalized reliability results using importance sampling

Description

This keyword controls how the probabilities at the MPP are computed: integration is followed by either `first_order` or `second_order`, indicating the order of the probability integration.

7.2.74.1.12.1 first_order

- [Keywords Area](#)
- [method](#)

- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [first_order](#)

First-order integration scheme

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

First-order integration in local reliability methods uses the minimum Euclidean distance from the origin to the most probable point (MPP) in transformed space to compute the probability of failure. This distance, commonly called the reliability index Beta, is used to calculate the probability of failure by calculating the standard normal cumulative distribution function at $-\text{Beta}$.

7.2.74.1.12.2 second_order

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [second_order](#)

Second-order integration scheme

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: none

Argument(s): none

Description

Second-order integration in local reliability methods modifies the first-order integration approach to apply a curvature correction. This correction is based on the formulation of [Hoh88 "Hohenbichler and Rackwitz, 1988"].

7.2.74.1.12.3 probability_refinement

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [probability_refinement](#)

Allow refinement of probability and generalized reliability results using importance sampling

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: sample_refinement

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Importance Sampling Approach (Group 1)	import	Importance sampling option for probability refinement
			adapt_import	Importance sampling option for probability refinement
			mm_adapt_import	Importance sampling option for probability refinement
	Optional		refinement_samples	Number of samples used to refine a probability estimate or sampling design.

	Optional	seed	Seed of the random number generator
--	-----------------	----------------------	-------------------------------------

Description

The `probability_refinement` allows refinement of probability and generalized reliability results using importance sampling. If one specifies `probability_refinement`, there are some additional options. One can specify which type of importance sampling to use (`import`, `adapt_import`, or `mm_adapt_import`). Additionally, one can specify the number of refinement samples to use with `refinement_samples` and the seed to use with `seed`.

The `probability_refinement` density reweighting accounts originally was developed based on Gaussian distributions. It now accounts for additional non-Gaussian cases.

7.2.74.1.12.4 `import`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [probability_refinement](#)
- [import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level).

7.2.74.1.12.5 `adapt_import`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [probability_refinement](#)
- [adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`adapt_import` centers a sampling density at one of the initial LHS samples identified in the failure region. It then generates the importance samples, weights them by their probability of occurrence given the original density, and calculates the required probability (CDF or CCDF level). This continues iteratively until the failure probability estimate converges.

7.2.74.1.12.6 `mm_adapt_import`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [probability_refinement](#)
- [mm_adapt_import](#)

Importance sampling option for probability refinement

Specification

Alias: none

Argument(s): none

Description

`mm_adapt_import` starts with all of the samples located in the failure region to build a multimodal sampling density. First, it uses a small number of samples around each of the initial samples in the failure region. Note that these samples are allocated to the different points based on their relative probabilities of occurrence: more probable points get more samples. This early part of the approach is done to search for "representative" points. Once these are located, the multimodal sampling density is set and then `mm_adapt_import` proceeds similarly to `adapt_import` (sample until convergence).

7.2.74.1.12.7 `refinement_samples`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [probability_refinement](#)
- [refinement_samples](#)

Number of samples used to refine a probability estimate or sampling design.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Probability estimate: Specify the (scalar) number of samples used to improve a probability estimate. If using uni-modal sampling all samples are assigned to the sampling center. If using multi-modal sampling the samples are split between multiple samples according to some internally computed weights.

Sampling design: Specify one or a sequence of refinement samples to augment the `initial_samples` in a sampling design.

7.2.74.1.12.8 `seed`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [mpp_search](#)
- [integration](#)
- [probability_refinement](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.74.2 response_levels

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing.

The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute_probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                  6.0e+04 6.5e+04 7.0e+04
                  3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors       = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas             = 250., 590.
  descriptors       = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors       = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors       = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note

that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12
3.6000000000e+11	4.0000000000e+11	4.2500000000e-12
4.0000000000e+11	4.4000000000e+11	3.7500000000e-12
4.4000000000e+11	5.4196114379e+11	2.2557612778e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	6.0000000000e+04	2.8742313192e-05
6.0000000000e+04	6.5000000000e+04	6.4000000000e-05
6.5000000000e+04	7.0000000000e+04	4.0000000000e-05
7.0000000000e+04	7.8702465755e+04	1.0341896485e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.5000000000e+05	4.2844660868e-06
3.5000000000e+05	4.0000000000e+05	8.6000000000e-06
4.0000000000e+05	4.5000000000e+05	1.8000000000e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
3.6000000000e+11	5.5000000000e-01		
4.0000000000e+11	3.8000000000e-01		
4.4000000000e+11	2.3000000000e-01		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
6.0000000000e+04	6.1000000000e-01		
6.5000000000e+04	2.9000000000e-01		
7.0000000000e+04	9.0000000000e-02		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
3.5000000000e+05	5.2000000000e-01		
4.0000000000e+05	9.0000000000e-02		
4.5000000000e+05	0.0000000000e+00		

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.74.2.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the `response_levels` will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.74.2.2 compute

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required (<i>Choose One</i>)	Statistics to Compute (Group 1)	probabilities	Computes probabilities associated with response levels
			reliabilities	Computes reliabilities associated with response levels
			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward statistical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, `probabilities` will be computed by default. If both `response_levels` and `compute` are specified, then one of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.74.2.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)

- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.74.2.2.2 reliabilities

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)
- [compute](#)
- [reliabilities](#)

Computes reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `reliabilities` keyword directs Dakota to compute reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the reliabilities are not computed by default. To change this behavior, the `reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.74.2.2.3 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

7.2.74.2.2.4 system

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group System Reliability Mode (Group 1)	Dakota Keyword series	Dakota Keyword Description Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.74.2.2.5 series

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.74.2.2.6 parallel

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.74.3 probability_levels

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_probability_levels	Specify which probability_levels correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors       = 'TF1n' 'TF2n'
```

```

uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
weibull_uncertain = 2
  alphas          = 12., 30.
  betas           = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
histogram_bin_uncertain = 2
  num_pairs       = 3      4
  abscissas       = 5 8 10 .1 .2 .3 .4
  counts          = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
histogram_point_uncertain
  real = 1
  num_pairs       = 2
  abscissas       = 3 4
  counts          = 1 1
  descriptors      = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		
4.0634975300e+11	3.3000000000e-01		
5.4196114379e+11	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_2:

Response Level	Probability Level	Reliability Index	General Rel Index
4.6431154744e+04	1.0000000000e+00		
5.6511827775e+04	8.0000000000e-01		
6.1603813790e+04	5.0000000000e-01		
7.8702465755e+04	0.0000000000e+00		

Complementary Cumulative Distribution Function (CCDF) for response_fn_3:

Response Level	Probability Level	Reliability Index	General Rel Index
2.3796737090e+05	1.0000000000e+00		

```
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.4111498127e+05  0.0000000000e+00
```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.74.3.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: probability_levels evenly distributed among response functions

Description

See parent page

7.2.74.4 reliability_levels

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [reliability_levels](#)

Specify reliability levels at which the response values will be estimated

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_reliability_levels	Specify which <code>reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF reliabilities by projecting out the prescribed number of sample standard deviations from the sample mean.

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.74.4.1 `num_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [reliability_levels](#)
- [num_reliability_levels](#)

Specify which `reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `reliability_levels` evenly distributed among response functions

Description

See parent page

7.2.74.5 gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which gen_reliability_levels correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.74.5.1 num_gen_reliability_levels

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability_levels evenly distributed among response functions

Description

See parent page

7.2.74.6 distribution

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.74.6.1 cumulative

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.74.6.2 complementary

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)

- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.74.7 max_iterations

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: `fsu_cvt` , `local_reliability`: 25; `global_{reliability , interval_est , evidence}` / `efficient_global`: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.74.8 `convergence_tolerance`

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.74.9 final_moments

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [final_moments](#)

Output moments of the specified type and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Default: standard

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Final moment type (Group 1)	none	Omit moments from the set of final statistics.
			standard	Output standardized moments and include them within the set of final statistics.

			central	Output central moments and include them within the set of final statistics.
--	--	--	-------------------------	---

Description

When performing a nested study that may employ moment statistics on the inner loop, it can be desirable to control the type of these moments. The `final_moments` specification supports options of `none`, `standard` (default), or `central`, corresponding to omission of moments, standardized moments (mean, standard deviation, skewness, and excess kurtosis), or central moments (mean, variance, 3rd central, and 4th central).

The presence or omission of moment results in the final statistics influences the outer level mappings in the case of a nested study. For example, `final_moments none` can allow for a more compact specification of primary and/or secondary response mappings.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

Overriding the default to `none` as follows:

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

allows associated nested model mappings to be simplified from:

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 0. 0. 1. 0. 0. 1. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 0. 0. 0. 0. 0. 1.
```

to a more compact version focused only on the response level mappings (two leading zeros per response function for moment mappings have been removed):

```
model
  nested
    sub_method_pointer = 'UQ'
    primary_response_mapping = 1. 1. 0.
    secondary_response_mapping = 0. 0. 1.
```

7.2.74.9.1 none

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [final_moments](#)

- [none](#)

Omit moments from the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

The omission of moment results from the final statistics can allow for a more compact definition of primary and/or secondary response mappings within a nested model specification.

Examples

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments none
```

7.2.74.9.2 standard

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [final_moments](#)
- [standard](#)

Output standardized moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output standardized moments (mean, standard deviation, skewness, and excess kurtosis) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic UQ). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specification.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments standard
```

7.2.74.9.3 central

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [final_moments](#)
- [central](#)

Output central moments and include them within the set of final statistics.

Specification

Alias: none

Argument(s): none

Description

Output central moments (mean, variance, 3rd central, and 4th central) and include the first two within the set of final statistics to be used at a higher level (e.g., optimization under uncertainty, mixed aleatory-epistemic U-Q). This inclusion of moment results affects the primary and/or secondary response mappings in a nested model specifier.

Examples

The following method specification overrides the default to print `central` moments and include them in the set of final statistics.

```
method,
  sampling
    samples = 50 seed = 1234
    response_levels = 3.6e+11 1.2e+05 3.5e+05
    final_moments central
```

7.2.74.10 model_pointer

- [Keywords Area](#)
- [method](#)
- [local_reliability](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.75 global_reliability

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)

Global reliability methods

Topics

This keyword is related to the topics:

- [uncertainty_quantification](#)
- [reliability_methods](#)

Specification

Alias: nond_global_reliability

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_samples	Initial number of samples for sampling-based methods
	Required (<i>Choose One</i>)	Approximation (Group 1)	x_gaussian_process	Create GP surrogate in x-space
			u_gaussian_process	Create GP surrogate in u-space
	Optional (<i>Choose One</i>)	GP Implementation (Group 2)	surfpack	Use the Surfpack version of Gaussian Process surrogates
			dakota	Select the built in Gaussian Process surrogate

			experimental	Use the experimental Gaussian Process surrogate
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations
	Optional		use_derivatives	Use derivative data to construct surrogate models
	Optional		seed	Seed of the random number generator
	Optional		rng	Selection of a random number generator
	Optional		response_levels	Values at which to estimate desired statistics for each response
	Optional		probability_levels	Specify probability levels at which to estimate the corresponding response value
	Optional		gen_reliability_levels	Specify generalized reliability levels at which to estimate the corresponding response value
	Optional		distribution	Selection of cumulative or complementary cumulative functions
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods

	Optional	convergence_ - tolerance	Stopping criterion based on objective function or statistics convergence
	Optional	model_pointer	Identifier for model block to be used by a method

Description

These methods do not support forward/inverse mappings involving `reliability_levels`, since they never form a reliability index based on distance in u-space. Rather they use a Gaussian process model to form an approximation to the limit state (based either in x-space via the `x_gaussian_process` specification or in u-space via the `u_gaussian_process` specification), followed by probability estimation based on multimodal adaptive importance sampling (see [10] and [11]). These probability estimates may then be transformed into generalized reliability levels if desired. At this time, inverse reliability analysis (mapping probability or generalized reliability levels into response levels) is not implemented.

The Gaussian process model approximation to the limit state is formed over the aleatory uncertain variables by default, but may be extended to also capture the effect of design, epistemic uncertain, and state variables. If this is desired, one must use the appropriate controls to specify the active variables in the variables specification block. Refer to [variable_support](#) for additional information on supported variable types.

See Also

These keywords may also be of interest:

- [adaptive_sampling](#)
- [gpais](#)
- [local_reliability](#)
- [sampling](#)
- [importance_sampling](#)
- [polynomial_chaos](#)
- [stoch_collocation](#)

7.2.75.1 initial_samples

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [initial_samples](#)

Initial number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: $(d+1)(d+2)/2$

Description

The `initial_samples` keyword is used to define the number of initial samples (i.e., randomly chosen sets of variable values) at which to execute a model. The initial samples may later be augmented in an iterative process.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10 \cdot \text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N \cdot (\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type random
    initial_samples = 20
    refinement_samples = 5
```

7.2.75.2 x_gaussian_process

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [x_gaussian_process](#)

Create GP surrogate in x-space

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: `x_kriging`

Argument(s): none

7.2.75.3 u_gaussian_process

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [u_gaussian_process](#)

Create GP surrogate in u-space

Topics

This keyword is related to the topics:

- [reliability_methods](#)

Specification

Alias: u_kriging

Argument(s): none

7.2.75.4 surfpack

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	Exports surrogate model in user-specified format(s)

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.2.75.4.1 `export_model`

- [Keywords Area](#)
- [method](#)

- [global_reliability](#)
- [surfpack](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)

- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.2.75.4.1.1 `filename_prefix`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [surfpack](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: `exported_surrogate`

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.2.75.4.1.2 formats

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See `export_model` and the entries for the format selection keywords for further information.

7.2.75.4.1.3 text_archive

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.75.4.1.4 binary_archive

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.75.5 dakota

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the `dakota` version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the `Surfpack` GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the Surfpack implementation is the default in all contexts except Bayesian calibration.

More details on the `gaussian_process` dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.2.75.6 experimental

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [experimental](#)

Use the experimental Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		options_file	Filename for a YAML file that specifies Gaussian process options

Description

The experimental GP surrogate is an emerging capability intended to supersede the `dakota` and `surfpack` GP implementations by unifying the best features of both. It also includes additional features such as the ability to export the model and then use it from Python.

See Also

These keywords may also be of interest:

- [experimental_gaussian_process](#)

7.2.75.6.1 export_model

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [experimental](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` surpack)
- Artificial neural network (keyword `neural_network`)
- Radial basis Funtions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the surpack executable with an exported model file.

7.2.75.6.1.1 filename_prefix

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [experimental](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: `exported_surrogate`

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.2.75.6.1.2 formats

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [experimental](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See `export_model` and the entries for the format selection keywords for further information.

7.2.75.6.1.3 text_archive

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.75.6.1.4 binary_archive

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [experimental](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.2.75.6.2 options_file

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [experimental](#)
- [options_file](#)

Filename for a YAML file that specifies Gaussian process options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the Gaussian process that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: gp_options.yaml

GP Parameters:
  Sigma Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  scaler name: standardization
  num restarts: 20
  Nugget:
    fixed nugget: 0.0
    estimate nugget: true
  Bounds:
    lower bound: 3.17e-8
    upper bound: 1.0e-2
  gp seed: 42
  standardize response: false
  Trend:
    estimate trend: true
  Options:
    max degree: 2
    reduced basis: false
    p-norm: 1.0
    scaler type: none
    regression solver type: SVD
    verbosity: 1
  kernel type: squared exponential
  Length-scale Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  verbosity: 1
```

7.2.75.7 import_build_points_file

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.2.75.7.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002      0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.75.7.1.1 header

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.2.75.7.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no eval_id column

Description

See description of parent `custom_annotated`

7.2.75.7.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.75.7.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.75.7.3 freeform

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857  0.2601620081  0.759955
0.89991          1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.75.7.4 active_only

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.75.8 `export_approx_points_file`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format

			freeform	Selects freeform file format
--	--	--	--------------------------	------------------------------

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.2.75.8.1 `custom_annotated`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file

	Optional	eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional	interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) annotated.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002          0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.2.75.8.1.1 header

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [export_approx_points_file](#)

- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent [custom_annotated](#)

7.2.75.8.1.2 [eval_id](#)

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no [eval_id](#) column

Description

See description of parent [custom_annotated](#)

7.2.75.8.1.3 [interface_id](#)

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no `interface_id` column

Description

See description of parent `custom_annotated`

7.2.75.8.2 annotated

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.75.8.3 freeform

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.75.9 use_derivatives

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surfpack Gaussian process).

However, its use with Surfpack Gaussian process is not recommended.

7.2.75.10 seed

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [seed](#)

Seed of the random number generator

Specification

Alias: none

Argument(s): INTEGER

Default: system-generated (non-repeatable)

Description

The random `seed` control provides a mechanism for making a stochastic method repeatable. That is, the use of the same random seed in identical studies will generate identical results.

Default Behavior

If not specified, the seed is randomly generated.

Expected Output

If `seed` is specified, a stochastic study will generate identical results when repeated using the same seed value. Otherwise, results are not guaranteed to be the same.

Usage Tips

If a stochastic study was run without `seed` specified, and the user later wishes to repeat the study using the same seed, the value of the seed used in the original study can be found in the output Dakota prints to the screen. That value can then be added to the Dakota input file.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 15347
```

7.2.75.11 rng

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [rng](#)

Selection of a random number generator

Specification

Alias: none

Argument(s): none

Default: Mersenne twister (mt19937)

Child Keywords:

	Required/Optional Required(<i>Choose One</i>)	Description of Group RNG Algorithm (Group 1)	Dakota Keyword mt19937	Dakota Keyword Description Generates random numbers using the Mersenne twister
			rnum2	Generates pseudo-random numbers using the Pecos package

Description

The `rng` keyword is used to indicate a choice of random number generator.

Default Behavior

If specified, the `rng` keyword must be accompanied by either `rnum2` (pseudo-random numbers) or `mt19937` (random numbers generated by the Mersenne twister). Otherwise, `mt19937`, the Mersenne twister is used by default.

Usage Tips

The default is recommended, as the Mersenne twister is a higher quality random number generator.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```

7.2.75.11.1 mt19937

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [rng](#)
- [mt19937](#)

Generates random numbers using the Mersenne twister

Specification

Alias: none

Argument(s): none

Description

The `mt19937` keyword directs Dakota to use the Mersenne twister to generate random numbers. Additional information can be found on wikipedia: http://en.wikipedia.org/wiki/Mersenne_twister.

Default Behavior

`mt19937` is the default random number generator. To specify it explicitly in the Dakota input file, however, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng mt19937
```

7.2.75.11.2 rnum2

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [rng](#)
- [rnum2](#)

Generates pseudo-random numbers using the Pecos package

Specification

Alias: none

Argument(s): none

Description

The `rnum2` keyword directs Dakota to use pseudo-random numbers generated by the Pecos package.

Default Behavior

`rnum2` is not used by default. To change this behavior, it must be specified in conjunction with the `rng` keyword.

Usage Tips

Use of the Mersenne twister random number generator (`mt19937`) is recommended over `rnum2`.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    seed = 98765
    rng rnum2
```


7.2.75.12 response_levels

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)

Values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF probabilities/reliabilities to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_response_levels	Number of values at which to estimate desired statistics for each response
	Optional		compute	Selection of statistics to compute at each response level

Description

The `response_levels` specification provides the target response values for which to compute probabilities, reliabilities, or generalized reliabilities (forward mapping).

Default Behavior

If `response_levels` are not specified, no statistics will be computed. If they are, probabilities will be computed by default.

Expected Outputs

If `response_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Usage Tips

The `num_response_levels` is used to specify which arguments of the `response_level` correspond to which response.

Examples

For example, specifying a `response_level` of 52.3 followed with `compute probabilities` will result in the calculation of the probability that the response value is less than or equal to 52.3, given the uncertain distributions on the inputs.

For an example with multiple responses, the following specification

```
response_levels = 1. 2. .1 .2 .3 .4 10. 20. 30.
num_response_levels = 2 4 3
```

would assign the first two response levels (1., 2.) to response function 1, the next four response levels (.1, .2, .3, .4) to response function 2, and the final three response levels (10., 20., 30.) to response function 3. If the `num_response_levels` key were omitted from this example, then the response levels would be evenly distributed among the response functions (three levels each in this case).

The Dakota input file below specifies a sampling method with response levels of interest.

```
method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  response_levels = 3.6e+11 4.0e+11 4.4e+11
                   6.0e+04 6.5e+04 7.0e+04
                   3.5e+05 4.0e+05 4.5e+05

variables,
  normal_uncertain = 2
  means            = 248.89, 593.33
  std_deviations   = 12.4, 29.7
  descriptors      = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds     = 199.3, 474.63
  upper_bounds     = 298.5, 712.
  descriptors      = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas           = 12., 30.
  betas            = 250., 590.
  descriptors      = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs        = 3 4
  abscissas        = 5 8 10 .1 .2 .3 .4
  counts           = 17 21 0 12 24 12 0
  descriptors      = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs        = 2
  abscissas        = 3 4
  counts           = 1 1
  descriptors      = 'TF3h'

interface,
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians
```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values specified in the input file. The probability levels corresponding to those response values are shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.6000000000e+11	5.3601733194e-12

```

3.6000000000e+11  4.0000000000e+11  4.2500000000e-12
4.0000000000e+11  4.4000000000e+11  3.7500000000e-12
4.4000000000e+11  5.4196114379e+11  2.2557612778e-12
PDF for response_fn_2:
  Bin Lower          Bin Upper          Density Value
-----
4.6431154744e+04  6.0000000000e+04  2.8742313192e-05
6.0000000000e+04  6.5000000000e+04  6.4000000000e-05
6.5000000000e+04  7.0000000000e+04  4.0000000000e-05
7.0000000000e+04  7.8702465755e+04  1.0341896485e-05
PDF for response_fn_3:
  Bin Lower          Bin Upper          Density Value
-----
2.3796737090e+05  3.5000000000e+05  4.2844660868e-06
3.5000000000e+05  4.0000000000e+05  8.6000000000e-06
4.0000000000e+05  4.5000000000e+05  1.8000000000e-06

```

Level mappings for each response function:

```

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
3.6000000000e+11  5.5000000000e-01
4.0000000000e+11  3.8000000000e-01
4.4000000000e+11  2.3000000000e-01
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
6.0000000000e+04  6.1000000000e-01
6.5000000000e+04  2.9000000000e-01
7.0000000000e+04  9.0000000000e-02
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
3.5000000000e+05  5.2000000000e-01
4.0000000000e+05  9.0000000000e-02
4.5000000000e+05  0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

A forward mapping involves computing the belief and plausibility probability level for a specified response level.

7.2.75.12.1 num_response_levels

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)
- [num_response_levels](#)

Number of values at which to estimate desired statistics for each response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: response_levels evenly distributed among response functions

Description

The `num_response_levels` keyword allows the user to specify the number of response values, for each response, at which estimated statistics are of interest. Statistics that can be computed are probabilities and reliabilities, both according to either a cumulative distribution function or a complementary cumulative distribution function.

Default Behavior

If `num_response_levels` is not specified, the response_levels will be evenly distributed among the responses.

Expected Outputs

The specific output will be determined by the type of statistics that are specified. In a general sense, the output will be a list of response level-statistic pairs that show the estimated value of the desired statistic for each response level specified.

Examples

```
method
  sampling
    samples = 100
    seed = 34785
    num_response_levels = 1 1 1
    response_levels = 0.5 0.5 0.5
```

7.2.75.12.2 compute

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)
- [compute](#)

Selection of statistics to compute at each response level

Specification

Alias: none

Argument(s): none

Default: probabilities

Child Keywords:

	Required/Optional	Description of Group Statistics to Compute (Group 1)	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)		probabilities	Computes probabilities associated with response levels

			gen_reliabilities	Computes generalized reliabilities associated with response levels
	Optional		system	Compute system reliability (series or parallel)

Description

The `compute` keyword is used to select which forward stastical mapping is calculated at each response level.

Default Behavior

If `response_levels` is not specified, no statistics are computed. If `response_levels` is specified but `compute` is not, probabilities will be computed by default. If both `response_levels` and `compute` are specified, then on of the following must be specified: `probabilities`, `reliabilities`, or `gen_reliabilities`.

Expected Output

The type of statistics specified by `compute` will be reported for each response level.

Usage Tips

CDF/CCDF probabilities are calculated for specified response levels using a simple binning approach.

CDF/CCDF reliabilities are calculated for specified response levels by computing the number of sample standard deviations separating the sample mean from the response level.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute reliabilities
```

7.2.75.12.2.1 probabilities

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)
- [compute](#)
- [probabilities](#)

Computes probabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `probabilities` keyword directs Dakota to compute the probability that the model response will be below (cumulative) or above (complementary cumulative) a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the probabilities are computed by default. To explicitly specify it in the Dakota input file, though, the `probabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-probability pairs that give the probability that the model response will be below or above the corresponding response level, depending on the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
    complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                    6.0e+04 6.5e+04 7.0e+04
                    3.5e+05 4.0e+05 4.5e+05
    compute probabilities
```

7.2.75.12.2.2 `gen_reliabilities`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)
- [compute](#)
- [gen_reliabilities](#)

Computes generalized reliabilities associated with response levels

Specification

Alias: none

Argument(s): none

Description

The `gen_reliabilities` keyword directs Dakota to compute generalized reliabilities according to the specified distribution for a specified response value. This is done for every response level designated for each response.

Default Behavior

If `response_levels` is specified, the generalized reliabilities are not computed by default. To change this behavior, the `gen_reliabilities` keyword should be specified in conjunction with the `compute` keyword.

Expected Outputs

The Dakota output is a set of response level-generalized reliability pairs according to the distribution defined.

Examples

```
method
  sampling
    sample_type random
    samples = 100 seed = 1
  complementary distribution
    response_levels = 3.6e+11 4.0e+11 4.4e+11
                      6.0e+04 6.5e+04 7.0e+04
                      3.5e+05 4.0e+05 4.5e+05
  compute gen_reliabilities
```

7.2.75.12.2.3 system

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)
- [compute](#)
- [system](#)

Compute system reliability (series or parallel)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	System Reliability Mode (Group 1)	series	Aggregate response statistics assuming a series system
			parallel	Aggregate response statistics assuming a parallel system

Description

With the system probability/reliability option, statistics for specified `response_levels` are calculated and reported assuming the response functions combine either in series or parallel to produce a total system response.

For a series system, the system fails when any one component (response) fails. The probability of failure is the complement of the product of the individual response success probabilities.

For a parallel system, the system fails only when all components (responses) fail. The probability of failure is the product of the individual response failure probabilities.

7.2.75.12.2.4 series

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [series](#)

Aggregate response statistics assuming a series system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.75.12.2.5 parallel

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [response_levels](#)
- [compute](#)
- [system](#)
- [parallel](#)

Aggregate response statistics assuming a parallel system

Specification

Alias: none

Argument(s): none

Description

See parent keyword `system` for description.

7.2.75.13 `probability_levels`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [probability_levels](#)

Specify probability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>num_probability_levels</code>	Specify which <code>probability_levels</code> correspond to which response

Description

Response levels are calculated for specified CDF/CCDF probabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Expected Output

If `probability_levels` are specified, Dakota will create two tables in the standard output: a Probability Density function (PDF) histogram and a Cumulative Distribution Function (CDF) table. The PDF histogram has the lower and upper endpoints of each bin and the corresponding density of that bin. Note that the PDF histogram has bins defined by the `probability_levels` and/or `response_levels` in the Dakota input file. If there are not very many levels, the histogram will be coarse. Dakota does not do anything to optimize the bin size or spacing. The CDF table has the list of response levels and the corresponding probability that the response value is less than or equal to each response level threshold.

Examples

The Dakota input file below specifies a sampling method with probability levels of interest.

```

method,
  sampling,
  samples = 100 seed = 1
  complementary distribution
  probability_levels = 1. .66 .33 0.
                    1. .8 .5 0.
                    1. .3 .2 0.

variables,
  normal_uncertain = 2
  means             = 248.89, 593.33
  std_deviations    = 12.4, 29.7
  descriptors        = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds      = 199.3, 474.63
  upper_bounds      = 298.5, 712.
  descriptors        = 'TF1u' 'TF2u'
  weibull_uncertain = 2
  alphas            = 12., 30.
  betas              = 250., 590.
  descriptors        = 'TF1w' 'TF2w'
  histogram_bin_uncertain = 2
  num_pairs         = 3 4
  abscissas         = 5 8 10 .1 .2 .3 .4
  counts            = 17 21 0 12 24 12 0
  descriptors        = 'TF1h' 'TF2h'
  histogram_point_uncertain
  real = 1
  num_pairs         = 2
  abscissas         = 3 4
  counts            = 1 1
  descriptors        = 'TF3h'

interface,
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses,
  response_functions = 3
  no_gradients
  no_hessians

```

Given the above Dakota input file, the following excerpt from the output shows the PDF and CCDF generated. Note that the bounds on the bins of the PDF are the response values that correspond the probability levels specified in the input file. Those response values are also shown in the CCDF.

Probability Density Function (PDF) histograms for each response function:

PDF for response_fn_1:

Bin Lower	Bin Upper	Density Value
2.7604749078e+11	3.4221494996e+11	5.1384774972e-12
3.4221494996e+11	4.0634975300e+11	5.1454122311e-12
4.0634975300e+11	5.4196114379e+11	2.4334239039e-12

PDF for response_fn_2:

Bin Lower	Bin Upper	Density Value
4.6431154744e+04	5.6511827775e+04	1.9839945149e-05
5.6511827775e+04	6.1603813790e+04	5.8916108390e-05
6.1603813790e+04	7.8702465755e+04	2.9242071306e-05

PDF for response_fn_3:

Bin Lower	Bin Upper	Density Value
2.3796737090e+05	3.6997214153e+05	5.3028386523e-06
3.6997214153e+05	3.8100966235e+05	9.0600055634e-06
3.8100966235e+05	4.4111498127e+05	3.3274925348e-06

Level mappings for each response function:

Complementary Cumulative Distribution Function (CCDF) for response_fn_1:

Response Level	Probability Level	Reliability Index	General Rel Index
2.7604749078e+11	1.0000000000e+00		
3.4221494996e+11	6.6000000000e-01		

```

4.0634975300e+11  3.3000000000e-01
5.4196114379e+11  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_2:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
4.6431154744e+04  1.0000000000e+00
5.6511827775e+04  8.0000000000e-01
6.1603813790e+04  5.0000000000e-01
7.8702465755e+04  0.0000000000e+00
Complementary Cumulative Distribution Function (CCDF) for response_fn_3:
  Response Level  Probability Level  Reliability Index  General Rel Index
-----
2.3796737090e+05  1.0000000000e+00
3.6997214153e+05  3.0000000000e-01
3.8100966235e+05  2.0000000000e-01
4.4111498127e+05  0.0000000000e+00

```

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.75.13.1 num_probability_levels

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [probability_levels](#)
- [num_probability_levels](#)

Specify which `probability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: `probability_levels` evenly distributed among response functions

Description

See parent page

7.2.75.14 gen_reliability_levels

- [Keywords Area](#)
- [method](#)

- [global_reliability](#)
- [gen_reliability_levels](#)

Specify generalized reliability levels at which to estimate the corresponding response value

Specification

Alias: none

Argument(s): REALLIST

Default: No CDF/CCDF response levels to compute

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_gen_reliability_levels	Specify which <code>gen_reliability_levels</code> correspond to which response

Description

Response levels are calculated for specified generalized reliabilities by indexing into a sorted samples array (the response levels computed are not interpolated and will correspond to one of the sampled values).

Theory

Sets of response-probability pairs computed with the forward/inverse mappings define either a cumulative distribution function (CDF) or a complementary cumulative distribution function (CCDF) for each response function.

In the case of evidence-based epistemic methods, this is generalized to define either cumulative belief and plausibility functions (CBF and CPF) or complementary cumulative belief and plausibility functions (CCBF and CCPF) for each response function.

An inverse mapping involves computing the belief and plausibility response level for either a specified probability level or a specified generalized reliability level (two results for each level mapping in the evidence-based epistemic case, instead of the one result for each level mapping in the aleatory case).

7.2.75.14.1 `num_gen_reliability_levels`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [gen_reliability_levels](#)
- [num_gen_reliability_levels](#)

Specify which `gen_reliability_levels` correspond to which response

Specification

Alias: none

Argument(s): INTEGERLIST

Default: gen_reliability_levels evenly distributed among response functions

Description

See parent page

7.2.75.15 distribution

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [distribution](#)

Selection of cumulative or complementary cumulative functions

Specification

Alias: none

Argument(s): none

Default: cumulative (CDF)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Distribution Type (CDF/CCDF) (Group 1)	cumulative	Computes statistics according to cumulative functions
			complementary	Computes statistics according to complementary cumulative functions

Description

The `distribution` keyword allows the user to select between a cumulative distribution/belief/plausibility function and a complementary cumulative distribution/belief/plausibility function. This choice affects how probabilities and reliability indices are reported.

Default Behavior

If the `distribution` keyword is present, it must be accompanied by either `cumulative` or `complementary`. Otherwise, a cumulative distribution will be used by default.

Expected Outputs

Output will be a set of model response-probability pairs determined according to the choice of distribution. The choice of distribution also defines the sign of the reliability or generalized reliability indices.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.75.15.1 cumulative

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [distribution](#)
- [cumulative](#)

Computes statistics according to cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a cumulative distribution/belief/plausibility function.

Default Behavior

By default, a cumulative distribution/belief/plausibility function will be used. To explicitly specify it in the Dakota input file, however, the `cumulative` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls below given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution cumulative
```

7.2.75.15.2 complementary

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)

- [distribution](#)
- [complementary](#)

Computes statistics according to complementary cumulative functions

Specification

Alias: none

Argument(s): none

Description

Statistics on model responses will be computed according to a complementary cumulative distribution/belief/plausibility function.

Default Behavior

By default, a complementary cumulative distribution/belief/plausibility function will not be used. To change that behavior, the `complementary` keyword must be appear in conjunction with the `distribution` keyword.

Expected Outputs

Output will be a set of model response-probability pairs determined according to a complementary cumulative distribution/belief/plausibility function. The probabilities reported are the probabilities that the model response falls above given response thresholds.

Examples

```
method
  sampling
    sample_type lhs
    samples = 10
    distribution complementary
```

7.2.75.16 max_iterations

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.75.17 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.75.18 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [global_reliability](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.76 fsu_quasi_mc

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)

Topics

This keyword is related to the topics:

- [package_fsudace](#)
- [design_and_analysis_of_computer_experiments](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Sequence Type (Group 1)	Dakota Keyword	Dakota Keyword Description
			halton	Generate samples from a Halton sequence
			hammersley	Use Hammersley sequences
	Optional		latinize	Adjust samples to improve the discrepancy of the marginal distributions
	Optional		quality_metrics	Calculate metrics to assess the quality of quasi-Monte Carlo samples
	Optional		variance_based_decomp	Activates global sensitivity analysis based on decomposition of response variance into contributions from variables
	Optional		samples	Number of samples for sampling-based methods
	Optional		fixed_sequence	Reuse the same sequence and samples for multiple sampling sets

	Optional	<code>sequence_start</code>	Choose where to start sampling the sequence
	Optional	<code>sequence_leap</code>	Specify how often the sequence is sampled
	Optional	<code>prime_base</code>	The prime numbers used to generate the sequence
	Optional	<code>max_iterations</code>	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional	<code>model_pointer</code>	Identifier for model block to be used by a method

Description

Quasi-Monte Carlo methods produce low discrepancy sequences, especially if one is interested in the uniformity of projections of the point sets onto lower dimensional faces of the hypercube (usually 1-D: how well do the marginal distributions approximate a uniform?)

This method generates sets of uniform random variables on the interval [0,1]. If the user specifies lower and upper bounds for a variable, the [0,1] samples are mapped to the [lower, upper] interval.

The user must first choose the sequence type:

- `halton` or
- `hammersley`

Then three keywords are used to define the sequence and how it is sampled:

- `prime_base`
- `sequence_start`
- `sequence_leap`

Each of these has defaults, so specification is optional.

Theory

The quasi-Monte Carlo sequences of Halton and Hammersley are deterministic sequences determined by a set of prime bases. Generally, we recommend that the user leave the default setting for the bases, which are the lowest primes. Thus, if one wants to generate a sample set for 3 random variables, the default bases used are 2, 3, and 5 in the Halton sequence. To give an example of how these sequences look, the Halton sequence in base 2 starts with points 0.5, 0.25, 0.75, 0.125, 0.625, etc. The first few points in a Halton base 3 sequence are 0.33333, 0.66667, 0.11111, 0.44444, 0.77777, etc. Notice that the Halton sequence tends to alternate back and forth, generating a point closer to zero then a point closer to one. An individual sequence is based on a radix inverse function defined on a prime base. The prime base determines how quickly the [0,1] interval is filled in. Generally, the lowest primes are recommended.

The Hammersley sequence is the same as the Halton sequence, except the values for the first random variable are equal to $1/N$, where N is the number of samples. Thus, if one wants to generate a sample set of 100 samples for 3

random variables, the first random variable has values $1/100$, $2/100$, $3/100$, etc. and the second and third variables are generated according to a Halton sequence with bases 2 and 3, respectively.

For more information about these sequences, see [44], [45], and [1].

See Also

These keywords may also be of interest:

- [dace](#)
- [fsu_cvt](#)
- [psuade_moat](#)

7.2.76.1 halton

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [halton](#)

Generate samples from a Halton sequence

Topics

This keyword is related to the topics:

- [package_fsudace](#)

Specification

Alias: none

Argument(s): none

Description

The quasi-Monte Carlo sequences of Halton are deterministic sequences determined by a set of prime bases. These sequences generate random numbers with the goal of filling a unit hypercube uniformly.

Generally, we recommend that the user leave the default setting for the bases, which are the lowest primes. Thus, if one wants to generate a sample set for 3 random variables, the default bases used are 2, 3, and 5 in the Halton sequence. To give an example of how these sequences look, the Halton sequence in base 2 starts with points 0.5, 0.25, 0.75, 0.125, 0.625, etc. The first few points in a Halton base 3 sequence are 0.33333, 0.66667, 0.11111, 0.44444, 0.77777, etc. Notice that the Halton sequence tends to alternate back and forth, generating a point closer to zero then a point closer to one. An individual sequence is based on a radix inverse function defined on a prime base. The prime base determines how quickly the $[0,1]$ interval is filled in.

Theory

For more information about these sequences, see [44], [45], and [1].

7.2.76.2 hammersley

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [hammersley](#)

Use Hammersley sequences

Topics

This keyword is related to the topics:

- [package_fsudace](#)
- [design_and_analysis_of_computer_experiments](#)

Specification

Alias: none

Argument(s): none

Description

The Hammersley sequence is the same as the Halton sequence, except the values for the first random variable are equal to $1/N$, where N is the number of samples. Thus, if one wants to generate a sample set of 100 samples for 3 random variables, the first random variable has values $1/100, 2/100, 3/100$, etc. and the second and third variables are generated according to a Halton sequence with bases 2 and 3, respectively.

See Also

These keywords may also be of interest:

- [fsu_quasi_mc](#)

7.2.76.3 latinize

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [latinize](#)

Adjust samples to improve the discrepancy of the marginal distributions

Specification

Alias: none

Argument(s): none

Default: No latinization

Description

The `latinize` control takes the samples and "latinizes" them, meaning that each original sample is moved so that it falls into one strata or bin in each dimension as in Latin Hypercube sampling. The default setting is NOT to latinize. However, one may be interested in doing this in situations where one wants better discrepancy of the 1-dimensional projections (the marginal distributions).

7.2.76.4 `quality_metrics`

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [quality_metrics](#)

Calculate metrics to assess the quality of quasi-Monte Carlo samples

Topics

This keyword is related to the topics:

- [package_fsudace](#)

Specification

Alias: none

Argument(s): none

Default: No `quality_metrics`

Description

`quality_metrics` calculates four quality metrics relating to the volumetric spacing of the samples. The four quality metrics measure different aspects relating to the uniformity of point samples in hypercubes. Desirable properties of such point samples are:

- are the points equally spaced
- do the points cover the region
- and are they isotropically distributed
- with no directional bias in the spacing

The four quality metrics we report are:

- `h`: the point distribution norm, which is a measure of uniformity of the point distribution
- `chi`: a regularity measure, and provides a measure of local uniformity of a set of points
- `tau`: the second moment trace measure
- `d`: the second moment determinant measure

All of these values are scaled so that smaller is better (the smaller the metric, the better the uniformity of the point distribution).

Examples

Complete explanation of these measures can be found in [39].

7.2.76.5 variance_based_decomp

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [variance_based_decomp](#)

Activates global sensitivity analysis based on decomposition of response variance into contributions from variables

Specification

Alias: none

Argument(s): none

Default: no variance-based decomposition

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		drop_tolerance	Suppresses output of sensitivity indices with values lower than this tolerance

Description

Dakota can calculate sensitivity indices through variance based decomposition using the keyword `variance_based_decomp`. These indicate how important the uncertainty in each input variable is in contributing to the output variance.

Default Behavior

Because of the computational cost, `variance_based_decomp` is turned off as a default.

If the user specified a number of samples, N, and a number of nondeterministic variables, M, variance-based decomposition requires the evaluation of $N*(M+2)$ samples. **Note that specifying this keyword will increase the number of function evaluations above the number requested with the `samples` keyword since replicated sets of sample values are evaluated.**

Expected Outputs

When `variance_based_decomp` is specified, sensitivity indices for main effects and total effects will be reported. Main effects (roughly) represent the percent contribution of each individual variable to the variance in the model response. Total effects represent the percent contribution of each individual variable in combination with all other variables to the variance in the model response

Usage Tips

To obtain sensitivity indices that are reasonably accurate, we recommend that N, the number of samples, be at least one hundred and preferably several hundred or thousands.

Examples

```
method,  
  sampling  
  sample_type lhs  
  samples = 100  
  variance_based_decomp
```

Theory

In this context, we take sensitivity analysis to be global, not local as when calculating derivatives of output variables with respect to input variables. Our definition is similar to that of [76] : "The study of how uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input."

Variance based decomposition is a way of using sets of samples to understand how the variance of the output behaves, with respect to each input variable. A larger value of the sensitivity index, S_i , means that the uncertainty in the input variable i has a larger effect on the variance of the output. More details on the calculations and interpretation of the sensitivity indices can be found in [76] and [90].

7.2.76.5.1 drop_tolerance

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [variance_based_decomp](#)
- [drop_tolerance](#)

Suppresses output of sensitivity indices with values lower than this tolerance

Specification

Alias: none

Argument(s): REAL

Default: All VBD indices displayed

Description

The `drop_tolerance` keyword allows the user to specify a value below which sensitivity indices generated by `variance_based_decomp` are not displayed.

Default Behavior

By default, all sensitivity indices generated by `variance_based_decomp` are displayed.

Usage Tips

For `polynomial_chaos`, which outputs main, interaction, and total effects by default, the `univariate_effects` may be a more appropriate option. It allows suppression of the interaction effects since the output volume of these results can be prohibitive for high dimensional problems. Similar to suppression of these interactions is the covariance control, which can be selected to be `diagonal_covariance` or `full_covariance`, with the former supporting suppression of the off-diagonal covariance terms (to save compute and memory resources and reduce output volume).

Examples

```
method,
  sampling
    sample_type lhs
    samples = 100
    variance_based_decomp
    drop_tolerance = 0.001
```

7.2.76.6 samples

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [samples](#)

Number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

The `samples` keyword is used to define the number of samples (i.e., randomly chosen sets of variable values) at which to execute a model.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10 \cdot \text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N \cdot (\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.2.76.7 fixed_sequence

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)

- [fixed_sequence](#)

Reuse the same sequence and samples for multiple sampling sets

Specification

Alias: none

Argument(s): none

Default: sequence not fixed: sampling patterns are variable among multiple QMC runs

Description

The `fixed_sequence` control is similar to `fixed_seed` for other sampling methods. If `fixed_sequence` is specified, the user will get the same sequence (meaning the same set of samples) for subsequent calls of the QMC sampling method (for example, this might be used in a surrogate based optimization method or a parameter study where one wants to fix the uncertain variables).

7.2.76.8 `sequence_start`

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [sequence_start](#)

Choose where to start sampling the sequence

Specification

Alias: none

Argument(s): INTEGERLIST

Default: Vector of zeroes

Description

`sequence_start` determines where in the sequence the samples will start.

The default `sequence_start` is a vector with 0 for each variable, specifying that each sequence start with the first term.

Examples

For example, for the Halton sequence in base 2, if the user specifies `sequence_start = 2`, the sequence would not include 0.5 and 0.25, but instead would start at 0.75.

7.2.76.9 `sequence_leap`

- [Keywords Area](#)
- [method](#)

- [fsu_quasi_mc](#)
- [sequence_leap](#)

Specify how often the sequence is sampled

Specification

Alias: none

Argument(s): INTEGERLIST

Default: Vector of ones

Description

`sequence_leap` controls the "leaping" of terms in the sequence. The default is 1 for each variable, meaning that each term in the sequence be returned.

Examples

If the user specifies a `sequence_leap` of 2 for a variable, the points returned would be every other term from the QMC sequence.

Theory

The advantage to using a leap value greater than one is mainly for high-dimensional sets of random deviates. In this case, setting a leap value to the next prime number larger than the largest prime base can help maintain uniformity when generating sample sets for high dimensions. For more information about the efficacy of leaped Halton sequences, see [74].

7.2.76.10 `prime_base`

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [prime_base](#)

The prime numbers used to generate the sequence

Specification

Alias: none

Argument(s): INTEGERLIST

Default: Vector of the first `s` primes for `s`-dimensions in Halton, First (`s-1`) primes for Hammersley

Description

It is recommended that the user not specify this and use the default values.

- For the Halton sequence, the default bases are primes in increasing order, starting with 2, 3, 5, etc.
- For the Hammersley sequence, the user specifies (s-1) primes if one is generating an s-dimensional set of random variables.

7.2.76.11 max_iterations

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.76.12 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [fsu_quasi_mc](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```
surrogate global,  
dace_method_pointer = 'DACE'  
polynomial quadratic  
  
method  
id_method = 'DACE'  
model_pointer = 'DACE_M'  
sampling sample_type lhs  
samples = 121 seed = 5034 rng rnum2  
  
model  
id_model = 'DACE_M'  
single  
interface_pointer = 'I1'  
  
variables  
uniform_uncertain = 2  
lower_bounds = 0. 0.  
upper_bounds = 1. 1.  
descriptors = 'x1' 'x2'  
  
interface  
id_interface = 'I1'  
system asynch evaluation_concurrency = 5  
analysis_driver = 'text_book'  
  
responses  
response_functions = 3  
no_gradients  
no_hessians
```

7.2.77 vector_parameter_study

- [Keywords Area](#)
- [method](#)
- [vector_parameter_study](#)

Samples variables along a user-defined vector

Topics

This keyword is related to the topics:

- [parameter_studies](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Step Control (Group 1)	final_point	Final variable values defining vector in vector parameter study
			step_vector	Size of step for each variable
	Required		num_steps	Number of sampling steps along the vector in a vector parameter study
	Optional		model_pointer	Identifier for model block to be used by a method

Description

Dakota's vector parameter study computes response data sets at selected intervals along a vector in parameter space. It is often used for single-coordinate parameter studies (to study the effect of a single variable on a response set), but it can be used more generally for multiple coordinate vector studies (to investigate the response variations along some n-dimensional vector such as an optimizer search direction).

Default Behavior

By default, the vector parameter study operates over all types of variables.

Expected Outputs

A vector parameter study produces a set of responses for each parameter set that is generated.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Parameter Sets](#)

Usage Tips

Group 1 is used to define the vector along which the parameters are varied. Both cases also rely on the variables specification of an initial value, through:

- the [initial_point](#) keyword
- the [initial_state](#) keyword
- relying on the default initial value, based on the rest of the variables specification

From the initial value, the vector can be defined using one of the two keyword choices.

Once the vector is defined, the samples are then fully specified by [num_steps](#).

Examples

The following example is a good comparison to the examples on [multidim_parameter_study](#) and [centered_parameter_study](#).


```
# tested on Dakota 6.0 on 140501
environment
  tabular_data
    tabular_data_file = 'rosen_vector.dat'

method
  vector_parameter_study
    num_steps = 10
    final_point = 2.0 2.0
model
  single

variables
  continuous_design = 2
  initial_point = -2.0 -2.0
  descriptors = 'x1' "x2"

interface
  analysis_driver = 'rosenbrock'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians
```

See Also

These keywords may also be of interest:

- [centered_parameter_study](#)
- [multidim_parameter_study](#)
- [list_parameter_study](#)

7.2.77.1 final_point

- [Keywords Area](#)
- [method](#)
- [vector_parameter_study](#)
- [final_point](#)

Final variable values defining vector in vector parameter study

Specification

Alias: none

Argument(s): REALLIST

Description

The `final_point` keyword is used to define the final values for each variable on the vector to be used in the vector parameter study. The vector's direction and magnitude are determined by the initial value from the variables specification, and the `final_point`.

Default Behavior

The user is required to specify either `final_point` or `step_vector`. There is no default definition for the vector.

Usage Tips

The actual points are determined based on this vector and the number of points chosen is given in `num_points`.

Examples

```
method
  vector_parameter_study
    num_steps = 10
    final_point = 2.0 2.0
```

7.2.77.2 `step_vector`

- [Keywords Area](#)
- [method](#)
- [vector_parameter_study](#)
- [step_vector](#)

Size of step for each variable

Specification

Alias: none

Argument(s): REALLIST

Description

The `step_vector` keyword specifies how much each variable will be incremented in a single step.

`step_vector` works in conjunction with `num_steps`, which determines the number of steps taken during the `vector_parameter_study`. If instead of `step_vector`, `final_point` is specified with `num_steps`, Dakota will infer the step sizes.

Entries in the `step_vector` are the actual amounts by which continuous and range variables are incremented. For set variables, `step_vector` entries are interpreted as indexes into the underlying set.

Default Behavior

The user is required to specify either `final_point` or `step_vector`. There is no default definition for the vector.

Examples

```
variables
  continuous_design 1
    initial_point 1.0
    descriptors 'x1'
  discrete_design_set
  string 1
    elements 'bar' 'baz' 'foo' 'fuzz'
    initial_point 'bar'
    descriptors 's1'

method
```

```
vector_parameter_study
  num_steps = 3
  # Add 2.0 to x1 and increment s1 by 1 element in each step
  step_vector = 2.0 1
```

7.2.77.3 num_steps

- [Keywords Area](#)
- [method](#)
- [vector_parameter_study](#)
- [num_steps](#)

Number of sampling steps along the vector in a vector parameter study

Specification

Alias: none

Argument(s): INTEGER

Description

`num_steps` defines the number of steps that are taken in the direction of the vector. The magnitude of each step is determined in conjunction with the rest of the method specification.

Default Behavior

The user is required to specify `num_steps` for a vector parameter study. There is no default value.

This study performs function evaluations at both ends, making the total number of evaluations equal to `num_steps+1`.

Usage Tips

The study has stringent requirements on performing appropriate steps with any discrete range and discrete set variables. A `num_steps` specification must result in discrete range and set index steps that are integers: no remainder is currently permitted in the integer step calculation and no rounding to integer steps will occur.

Examples

```
method
  vector_parameter_study
    num_steps = 10
    final_point = 2.0 2.0
```

7.2.77.4 model_pointer

- [Keywords Area](#)
- [method](#)
- [vector_parameter_study](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'
```

```

variables
  uniform_uncertain = 2
  lower_bounds = 0.  0.
  upper_bounds = 1.  1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system_async_evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians

```

7.2.78 list_parameter_study

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)

Samples variables as a specified values

Topics

This keyword is related to the topics:

- [parameter_studies](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Points Source (Group 1)	Dakota Keyword list_of_points	Dakota Keyword Description List of variable values to evaluate in a list parameter study
			import_points_file	File containing list of variable values to evaluate in a list parameter study

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

Dakota's list parameter study allows for evaluations at user selected points of interest.

Default Behavior

By default, the list parameter study operates over all types of variables.

The number of real values in the `list_of_points` specification or file referenced by `import_points_file` must be a multiple of the total number of variables (including continuous and discrete types) contained in the variables specification.

Expected Outputs

A list parameter study produces a set of responses for each parameter set that is specified.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Parameter Sets](#)

Usage Tips

- This parameter study simply performs simulations for the first parameter set (the first n entries in the list), followed by the next parameter set (the next n entries), and so on, until the list of points has been exhausted.
- Since the initial values from the variables specification will not be used, they need not be specified.
- When the points are specified in the Dakota input file using the `list_of_points` keyword, discrete set values must be referred to using 0-based indexes, and not the values themselves. However, when using `import_points_file`, refer to discrete set values directly, not by index.
- For both `list_of_points` and `import_points_file`, Dakota expects the values for each evaluation to be ordered by type. The type ordering matches that of the [variables](#) section of this Reference Manual and of the listing in the Parameters file format section of the Dakota User's Manual [5]. When multiple variables are present for a single type, the ordering within that type must match the order specified by the user in the variables section of her input file.

Examples

This shows the method and variables block of a Dakota input file that runs a `list_parameter_study`.

```
method
  list_parameter_study
  list_of_points =
    3.1e6    0.0029    0.31
    3.2e6    0.0028    0.32
    3.3e6    0.0027    0.34
    3.3e6    0.0026    0.36

variables
  continuous_design = 3
  descriptors = 'E'   'MASS'   'DENSITY'
```

Note that because of the way Dakota treats whitespace, the above example is equivalent to:

```

method
  list_parameter_study
  list_of_points =
3.1e6   0.0029   0.31   3.2e6   0.0028
0.32   3.3e6   0.0027
0.34  3.3e6   0.0026   0.36

variables
  continuous_design = 3
  descriptors = 'E' 'MASS' 'DENSITY'

```

Although the first example is much more readable.

And here's a full input file:

```

# tested on Dakota 6.0 on 140501
environment
  tabular_data
  tabular_data_file 'List_param_study.dat'

method
  list_parameter_study
  list_of_points =
      0.1   0.1
      0.2   0.1
      0.3   0.0
      0.3   1.0

model
  single

variables
  active design
  continuous_design = 2
  descriptors      'x1' 'x2'
  continuous_state = 1
  descriptors = 'constant1'
  initial_state = 100

interface
  analysis_drivers 'text_book'
  fork
  asynchronous
  evaluation_concurrency 2

responses
  response_functions = 1
  no_gradients
  no_hessians

```

This example illustrates the `list_parameter_study`.

- The function evaluations are independent, so any level of `evaluation_concurrency` can be used
- Default behavior for parameter studies is to iterate on all variables. However, because `active design` is specified, this study will only iterate on the `continuous_design` variables.

See Also

These keywords may also be of interest:

- [centered_parameter_study](#)
- [multidim_parameter_study](#)
- [vector_parameter_study](#)

7.2.78.1 list_of_points

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [list_of_points](#)

List of variable values to evaluate in a list parameter study

Specification

Alias: none

Argument(s): REALLIST

Description

The `list_of_points` keyword allows the user to specify, in a freeform format, a list of variable values at which to compute a model response.

Default Behavior

The user is required to provide a list of points for a list parameter study either by specifying it with `list_of_points` or by providing a file from which such a list can be read via `import_points_file`. There is no default list of points.

Usage Tips

The number of values in the list must be an integer multiple of the number of variables. Dakota will verify that this condition is met.

Examples

```
method
  list_parameter_study
    list_of_points =
      3.1e6    0.0029    0.31
      3.2e6    0.0028    0.32
      3.3e6    0.0027    0.34
      3.3e6    0.0026    0.36
```

7.2.78.2 import_points_file

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)

File containing list of variable values to evaluate in a list parameter study

Specification

Alias: none

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_points_file` specifies a file containing a list of variable values at which to compute a model response.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_points_file` must be in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

Examples

```
method
  list_parameter_study
  import_points_file = 'dakota_pstudy.3.dat'
```

7.2.78.2.1 custom_annotated

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1      x2      obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9      1.1      0.0002      0.26      0.76
2              0.90009  1.1 0.0001996404857  0.2601620081  0.759955
3              0.89991  1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.2.78.2.1.1 header

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.78.2.1.2 eval_id

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.78.2.1.3 `interface_id`

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.2.78.2.2 `annotated`

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.78.2.3 freeform

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.2.78.2.4 active_only

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [import_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.2.78.3 `model_pointer`

- [Keywords Area](#)
- [method](#)
- [list_parameter_study](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.2.79 centered_parameter_study

- [Keywords Area](#)
- [method](#)
- [centered_parameter_study](#)

Samples variables along points moving out from a center point

Topics

This keyword is related to the topics:

- [parameter_studies](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		step_vector	Size of steps to be taken in each dimension of a centered parameter study
	Required		steps_per_variable	Number of steps to take in each dimension of a centered parameter study
	Optional		model_pointer	Identifier for model block to be used by a method

Description

Dakota's centered parameter study computes response data sets along multiple coordinate-based vectors, one per parameter, centered about the initial values from the variables specification. This is useful for investigation of function contours with respect to each parameter individually in the vicinity of a specific point (e.g., post-optimality analysis for verification of a minimum), thereby avoiding the cost associated with a multidimensional grid.

Default Behavior

By default, the centered parameter study operates over all types of variables.

The `centered_parameter_study` takes steps along each orthogonal dimension. Each dimension is treated independently. The number of steps are taken in each direction, so that the total number of points in the parameter study is $1 + 2 \sum n$.

Expected Outputs

A centered parameter study produces a set of responses for each parameter set that is generated.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the `hdf5` keyword, this method writes the following results to HDF5:

- [Parameter Sets](#)
- [Variable Slices](#)

Examples

The following example is a good comparison to the examples on [multidim_parameter_study](#) and [vector_parameter_study](#).

```
# tested on Dakota 6.0 on 140501
environment
  tabular_data
    tabular_data_file = 'rosen_centered.dat'

method
  centered_parameter_study
    steps_per_variable = 5 4
    step_vector = 0.4 0.5

model
  single

variables
  continuous_design = 2
  initial_point = 0      0
  descriptors = 'x1'    "x2"

interface
  analysis_driver = 'rosenbrock'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians
```

See Also

These keywords may also be of interest:

- [multidim_parameter_study](#)
- [list_parameter_study](#)
- [vector_parameter_study](#)

7.2.79.1 step_vector

- [Keywords Area](#)
- [method](#)
- [centered_parameter_study](#)
- [step_vector](#)

Size of steps to be taken in each dimension of a centered parameter study

Specification

Alias: none

Argument(s): REALLIST

Description

The `step_vector` keyword defines the individual step size in each dimension, treated separately.

Default Behavior

The user is required to define the number of step sizes for a centered parameter study. There are no default values.

Steps are taken in the plus and minus directions, and are defined in either actual values (continuous and discrete range) or index offsets (discrete set).

Examples

```
method
  centered_parameter_study
    steps_per_variable = 5 4
    step_vector = 0.4 0.5
```

7.2.79.2 steps_per_variable

- [Keywords Area](#)
- [method](#)
- [centered_parameter_study](#)
- [steps_per_variable](#)

Number of steps to take in each dimension of a centered parameter study

Specification

Alias: `deltas_per_variable`

Argument(s): INTEGERLIST

Description

The `steps_per_variable` keyword allows the user to define the number of steps in each dimension of a centered parameter study. Because they are taken independently, the number of steps can be specified for each.

Default Behavior

The user is required to define the number of steps per variable for a centered parameter study. There are no default values.

Steps are taken in the plus and minus directions, and are defined in either actual values (continuous and discrete range) or index offsets (discrete set).

Examples

```
method
  centered_parameter_study
    steps_per_variable = 5 4
    step_vector = 0.4 0.5
```

7.2.79.3 model_pointer

- [Keywords Area](#)
- [method](#)
- [centered_parameter_study](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
  seed = 98765 rng rnum2
  response_levels = 0.1 0.2 0.6
                   0.1 0.2 0.6
                   0.1 0.2 0.6

  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
```

```

surrogate global,
dace_method_pointer = 'DACE'
polynomial quadratic

method
id_method = 'DACE'
model_pointer = 'DACE_M'
sampling sample_type lhs
samples = 121 seed = 5034 rng rnum2

model
id_model = 'DACE_M'
single
interface_pointer = 'I1'

variables
uniform_uncertain = 2
lower_bounds = 0. 0.
upper_bounds = 1. 1.
descriptors = 'x1' 'x2'

interface
id_interface = 'I1'
system async evaluation_concurrency = 5
analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.80 multidim_parameter_study

- [Keywords Area](#)
- [method](#)
- [multidim_parameter_study](#)

Samples variables on full factorial grid of study points

Topics

This keyword is related to the topics:

- [parameter_studies](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		partitions	Samples variables on full factorial grid of study points

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

Dakota's multidimensional parameter study computes response data sets for an n-dimensional grid of points. Each continuous and discrete range variable is partitioned into equally spaced intervals between its upper and lower bounds, each discrete set variable is partitioned into equally spaced index intervals. The partition boundaries in n-dimensional space define a grid of points, and every point is evaluated.

Default Behavior

By default, the multidimensional parameter study operates over all types of variables.

Expected Outputs

A multidimensional parameter study produces a set of responses for each parameter set that is generated.

Expected HDF5 Output

If Dakota was built with HDF5 support and run with the [hdf5](#) keyword, this method writes the following results to HDF5:

- [Parameter Sets](#)
- [Correlations](#)

Usage Tips

Since the initial values from the variables specification will not be used, they need not be specified.

Examples

This example is taken from the Users Manual and is a good comparison to the examples on [centered_parameter_study](#) and [vector_parameter_study](#).

```
# tested on Dakota 6.0 on 140501
environment
  tabular_data
    tabular_data_file = 'rosen_multidim.dat'

method
  multidim_parameter_study
    partitions = 10 8

model
  single

variables
  continuous_design = 2
  lower_bounds      -2.0   -2.0
  upper_bounds      2.0    2.0
  descriptors        'x1'   "x2"

interface
  analysis_driver = 'rosenbrock'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians
```

This example illustrates the full factorial combinations of parameter values created by the `multidim_parameter_study`. With 10 and 8 partitions, there are actually 11 and 9 values for each variable. This means that $11 \times 9 = 99$ function evaluations will be required.

See Also

These keywords may also be of interest:

- [centered_parameter_study](#)
- [list_parameter_study](#)
- [vector_parameter_study](#)

7.2.80.1 partitions

- [Keywords Area](#)
- [method](#)
- [multidim_parameter_study](#)
- [partitions](#)

Samples variables on full factorial grid of study points

Topics

This keyword is related to the topics:

- [parameter_studies](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Dakota's multidimensional parameter study computes response data sets for an n-dimensional grid of points. Each continuous and discrete range variable is partitioned into equally spaced intervals between its upper and lower bounds, each discrete set variable is partitioned into equally spaced index intervals. The partition boundaries in n-dimensional space define a grid of points, and every point is evaluated.

Default Behavior

By default, the multidimensional parameter study operates over all types of variables.

Expected Outputs

A multidimensional parameter study produces a set of responses for each parameter set that is generated.

Usage Tips

Since the initial values from the variables specification will not be used, they need not be specified.

Examples

This example is taken from the Users Manual and is a good comparison to the examples on [centered_parameter_study](#) and [vector_parameter_study](#).

```
# tested on Dakota 6.0 on 140501
environment
  tabular_data
    tabular_data_file = 'rosen_multidim.dat'

method
  multidim_parameter_study
    partitions = 10 8

model
  single

variables
  continuous_design = 2
  lower_bounds      -2.0      -2.0
  upper_bounds      2.0       2.0
  descriptors       'x1'      "x2"

interface
  analysis_driver = 'rosenbrock'
  fork

responses
  response_functions = 1
  no_gradients
  no_hessians
```

This example illustrates the full factorial combinations of parameter values created by the `multidim_parameter_study`. With 10 and 8 partitions, there are actually 11 and 9 values for each variable. This means that $11 \times 9 = 99$ function evaluations will be required.

See Also

These keywords may also be of interest:

- [centered_parameter_study](#)
- [list_parameter_study](#)
- [vector_parameter_study](#)

7.2.80.2 model_pointer

- [Keywords Area](#)
- [method](#)
- [multidim_parameter_study](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which `model` block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a `model` block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'

method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
    samples = 10
    seed = 98765 rng rnum2
    response_levels = 0.1 0.2 0.6
                    0.1 0.2 0.6
                    0.1 0.2 0.6
  sample_type lhs
  distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0. 0.
  upper_bounds = 1. 1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system async evaluation_concurrency = 5
```

```

analysis_driver = 'text_book'

responses
response_functions = 3
no_gradients
no_hessians

```

7.2.81 richardson_extrap

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)

Estimate order of convergence of a response as model fidelity increases

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Verification Mode (Group 1)	estimate_order	Compute the best estimate of the convergence order from three points
			converge_order	Refine until the estimated convergence order converges
			converge_qoi	Refine until the response converges
	Optional		refinement_rate	Rate at which the state variables are refined
	Optional		convergence_tolerance	Stopping criterion based on objective function or statistics convergence
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods

	Optional	model_pointer	Identifier for model block to be used by a method
--	-----------------	-------------------------------	---

Description

Solution verification procedures estimate the order of convergence of the simulation response data during the course of a refinement study. This branch of methods is new and currently only contains one algorithm: Richardson extrapolation.

Refinement of the model

The model fidelity must be parameterized by one or more continuous state variable(s).

The refinement path is determined from the `initial_state` of the `continuous_state` variables specification in combination with the `refinement_rate`, where each of the state variables is treated as an independent refinement factor and each of the initial state values is repeatedly divided by the refinement rate value to define new discretization states.

Results

Three algorithm options are currently provided:

1. `estimate_order`
2. `converge_order`
3. `converge_qoi`

Stopping Criteria

The method employs the `max_iterations` and `convergence_tolerance` method independent controls as stopping criteria.

Theory

In each of these cases, convergence order for a response quantity of interest (QoI) is estimated from

$$p = \ln \left(\frac{QoI_3 - QoI_2}{QoI_2 - QoI_1} \right) / \ln(r)$$

where r is the uniform refinement rate specified by `refinement_rate`.

7.2.81.1 `estimate_order`

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)
- [estimate_order](#)

Compute the best estimate of the convergence order from three points

Specification

Alias: none

Argument(s): none

Description

The `estimate_order` option is the simplest option. For each of the refinement factors, it evaluates three points along the refinement path and uses these results to perform an estimate of the convergence order for each response function.

7.2.81.2 `converge_order`

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)
- [converge_order](#)

Refine until the estimated coverage order converges

Specification

Alias: none

Argument(s): none

Description

The `converge_order` option is initialized using the `estimate_order` approach, and additional refinements are performed along the refinement path until the convergence order estimates converge (two-norm of the change in response orders is less than the convergence tolerance).

7.2.81.3 `converge_qoi`

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)
- [converge_qoi](#)

Refine until the response converges

Specification

Alias: none

Argument(s): none

Description

The `converge_qoi` option is similar to the `converge_order` option, except that the convergence criterion is that the two-norm of the response discretization errors (computed from extrapolation) must be less than the convergence tolerance.

7.2.81.4 refinement_rate

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)
- [refinement_rate](#)

Rate at which the state variables are refined

Specification

Alias: none

Argument(s): REAL

Default: 2.

Description

Described on parent page

7.2.81.5 convergence_tolerance

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)
- [convergence_tolerance](#)

Stopping criterion based on objective function or statistics convergence

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): REAL

Default: 1.e-4

Description

The `convergence_tolerance` specification provides a real value for controlling the termination of iteration.

For optimization, it is most commonly a **relative convergence tolerance** for the objective function; i.e., if the change in the objective function between successive iterations divided by the previous objective function is less than the amount specified by `convergence_tolerance`, then this convergence criterion is satisfied on the current iteration.

Therefore, permissible values are between 0 and 1, non-inclusive.

Behavior Varies by Package/Library

This control is used with most optimization and least squares iterators (DOT, CONMIN, NLPQLP, NPSOL, NLSSOL, OPT++, and SCOLIB). Most other Dakota methods (such as DACE or parameter studies) do not use this control, but some adaptive methods, such as adaptive UQ, do.

Since no progress may be made on one iteration followed by significant progress on a subsequent iteration, some libraries require that the convergence tolerance be satisfied on two or more consecutive iterations prior to termination of iteration.

Notes on each library:

- DOT: relative tolerance that must be satisfied for two consecutive iterations
- NL2SOL: See [nl2sol](#)
- NLPQLP: used as Lagrangian gradient norm tolerance (ACC), not as a relative convergence tolerance
- NPSOL: used as a line search tolerance, not as a relative convergence tolerance

7.2.81.6 max_iterations

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100 (exceptions: fsu_cvt , local_reliability: 25; global_{reliability , interval_est , evidence} / efficient_global: 25*n)

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.2.81.7 model_pointer

- [Keywords Area](#)
- [method](#)
- [richardson_extrap](#)
- [model_pointer](#)

Identifier for model block to be used by a method

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed (or use of default model if none parsed)

Description

The `model_pointer` is used to specify which [model](#) block will be used to perform the function evaluations needed by the Dakota method.

Default Behavior

If not specified, a Dakota method will use the last model block parsed. If specified, there must be a [model](#) block in the Dakota input file that has a corresponding `id_model` with the same name.

Usage Tips

When doing advanced analyses that involve using multiple methods and multiple models, defining a `model_pointer` for each method is imperative.

See [block_pointer](#) for details about pointers.

Examples

```
environment
  tabular_data
  method_pointer = 'UQ'
```

```
method
  id_method = 'UQ'
  model_pointer = 'SURR'
  sampling,
  samples = 10
```

```
seed = 98765 rng rnum2
response_levels = 0.1 0.2 0.6
                 0.1 0.2 0.6
                 0.1 0.2 0.6

sample_type lhs
distribution cumulative

model
  id_model = 'SURR'
  surrogate global,
  dace_method_pointer = 'DACE'
  polynomial quadratic

method
  id_method = 'DACE'
  model_pointer = 'DACE_M'
  sampling sample_type lhs
  samples = 121 seed = 5034 rng rnum2

model
  id_model = 'DACE_M'
  single
  interface_pointer = 'I1'

variables
  uniform_uncertain = 2
  lower_bounds = 0.  0.
  upper_bounds = 1.  1.
  descriptors = 'x1' 'x2'

interface
  id_interface = 'I1'
  system asynch evaluation_concurrency = 5
  analysis_driver = 'text_book'

responses
  response_functions = 3
  no_gradients
  no_hessians
```

7.3 model

- [Keywords Area](#)
- [model](#)

Specifies how variables are mapped into a set of responses

Topics

This keyword is related to the topics:

- [block](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		id_model	Give the model block an identifying name, in case of multiple model blocks
	Required (<i>Choose One</i>)	Model Type (Group 1)	single	A model with one of each block: variable, interface, and response
			surrogate	An empirical model that is created from data or the results of a submodel
			nested	A model whose responses are computed through the use of a sub-iterator
			active_subspace	Active (variable) subspace model
			adapted_basis	Basis adaptation model
			random_field	Experimental capability to generate a random field representation. from data, from simulation runs, or from a covariance matrix. The representation may then be sampled for use as a random field input to another simulation. THIS IS AN EXPERIMENTAL CAPABILITY.

	Optional	<code>variables_pointer</code>	Specify which variables block will be included with this model block
	Optional	<code>responses_pointer</code>	Specify which responses block will be used by this model block
	Optional	<code>hierarchical_tagging</code>	Enables hierarchical evaluation tagging

Description

A model is comprised of a mapping from variables, through an interface, to responses.

Model Group 1 The type of model can be:

1. `single` (default)
2. `surrogate`
3. `nested`
4. `subspace`
5. `random_field`

The input file must specify one of these types. If the type is not specified, Dakota will assume a single model.

Block Pointers and ID

Each of these model types supports `variables_pointer` and `responses_pointer` strings for identifying the variables and responses specifications used in constructing the model by cross-referencing with `id_variables` and `id_responses` strings from particular variables and responses keyword specifications.

These pointers are valid for each model type since each model contains a set of variables that is mapped into a set of responses – only the specifics of the mapping differ.

Additional pointers are used for each model type for constructing the components of the variable to response mapping. As an environment specification identifies a top-level method and a method specification identifies a model, a model specification identifies variables, responses, and (for some types) interface specifications. This top-down flow specifies all of the object interrelationships.

Examples

The first example shows a minimal specification for a `single` model, which is the default model when no models are explicitly specified by the user.

```
model
  single
```

The next example displays a surrogate model specification which selects a quadratic polynomial from among the global approximation methods. It uses a pointer to a design of experiments method for generating the data needed for building the global approximation, reuses any old data available for the current approximation region, and employs the first-order multiplicative approach to correcting the approximation each time correction is requested.

```
model,
  id_model = 'M1'
  variables_pointer = 'V1'
  responses_pointer = 'R1'
```

```

surrogate
  global
    polynomial quadratic
    dace_method_pointer = 'DACE'
    reuse_samples region
    correction multiplicative first_order

```

This example demonstrates the use of identifiers and pointers. It provides the optional model independent specifications for model identifier, variables pointer, and responses pointer as well as model dependent specifications for global surrogates (see [global](#)).

Finally, an advanced nested model example would be

```

model
  id_model = 'M1'
  variables_pointer = 'V1'
  responses_pointer = 'R1'
  nested
    optional_interface_pointer = 'OI1'
    optional_interface_responses_pointer = 'OIR1'
    sub_method_pointer = 'SM1'
    primary_variable_mapping = '' '' 'X' 'Y'
    secondary_variable_mapping = '' '' 'mean' 'mean'
    primary_response_mapping = 1. 0. 0. 0. 0. 0. 0. 0. 0.
    secondary_response_mapping = 0. 0. 0. 1. 3. 0. 0. 0. 0.
                                0. 0. 0. 0. 0. 0. 1. 3. 0.

```

This example illustrates controls for model identifier, variables pointer, and responses pointer and for specifying details of the nested mapping.

7.3.1 id_model

- [Keywords Area](#)
- [model](#)
- [id_model](#)

Give the model block an identifying name, in case of multiple model blocks

Topics

This keyword is related to the topics:

- [block_identifier](#)

Specification

Alias: none

Argument(s): STRING

Default: method use of last model parsed

Description

The model identifier string is supplied with `id_model` and is used to provide a unique identifier string for use within method specifications (refer to the keyword `model_pointer` in any of the methods in the [method](#) block, for example: [model_pointer](#))

This is used to determine which model the method will run.

See Also

These keywords may also be of interest:

- [model_pointer](#)

7.3.2 single

- [Keywords Area](#)
- [model](#)
- [single](#)

A model with one of each block: variable, interface, and response

Specification

Alias: simulation

Argument(s): none

Default: N/A (single if no model specification)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		interface_pointer	Interface block pointer for the single model type
	Optional		solution_level_-control	Cost estimates associated with a set of solution control values.
	Optional (<i>Choose One</i>)	Cost model for 1 or more resolution levels (Group 1)	solution_level_cost	Cost estimates associated with a set of solution control values.
			cost_recovery_-metadata	Identify metadata by label for capturing online cost estimates

Description

The single model is the simplest model type. It uses a single [interface](#) instance to map [variables](#) into [responses](#). There is no recursion in this case.

The optional [interface_pointer](#) specification identifies the interface block by cross-referencing with the `id_interface` string input from a particular interface keyword specification. This is only necessary when the input file has multiple interface blocks, and you wish to explicitly point to the desired block. The same logic follows for responses and variables blocks and pointers.

Examples

The example shows a minimal specification for a single model, which is the default model when no models are specified by the user.

```
model
  single
```

This example does not provide any pointer strings and therefore relies on the default behavior of constructing the model with the last variables, interface, and responses specifications parsed.

See Also

These keywords may also be of interest:

- [surrogate](#)
- [nested](#)

7.3.2.1 interface_pointer

- [Keywords Area](#)
- [model](#)
- [single](#)
- [interface_pointer](#)

Interface block pointer for the single model type

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: model use of last interface parsed

Description

In the `single` model case, a single interface is used to map the variables into responses. The optional `interface_pointer` specification identifies this interface by cross-referencing with the `id_interface` string input from a particular interface keyword specification.

See [block_pointer](#) for details about pointers.

When an interface pointer is not specified, the model will use the last interface block parsed from the input file.

7.3.2.2 solution_level_control

- [Keywords Area](#)
- [model](#)
- [single](#)
- [solution_level_control](#)

Cost estimates associated with a set of solution control values.

Specification

Alias: none

Argument(s): STRING

Default: use of single default solution level

Description

Simulation-based models may have an associated `solution_level_control`, which identifies a hierarchy of solution states, such as a set of mesh discretizations from coarse to fine, a set of solver tolerances from loose to tight, etc. The string specified as the `solution_level_control` identifies a discrete variable label that parameterizes the hierarchy of solution states.

Note: If the discrete variable identified is a discrete set variable, then it is important to note that the variable's set values will be ordered (lexicographically in the case of string variables), and the ordering of values provided in `solution_level_cost` should correspond to this set ordering. A common error is to provide a listing of set values that is out of order and then providing a set of costs corresponding to this misordered list – in this case, the solution level costs will be associated with the re-ordered set values.

Examples

In this example, integer solution control values and solution costs are naturally well ordered.

```
model,
  simulation
    solution_level_control = 'mesh_size'
    solution_level_cost = 1. 8. 64. 512. 4096.

variables,
  uniform_uncertain = 9
  lower_bounds      = 9*-1.
  upper_bounds      = 9* 1.
  discrete_state_set
    integer = 1
    set_values = 4 8 16 32 64
    descriptors = 'mesh_size'
```

In this example, string solution control values are lexicographically ordered, and care must be taken to align the solution cost estimates.

```
model,
  simulation
    solution_level_control = 'mesh_size'
    solution_level_cost = 1. 64. 8. # match set ordering

variables,
  uniform_uncertain = 9
  lower_bounds      = 9*-1.
  upper_bounds      = 9* 1.
```

```

discrete_state_set
  string = 1
    set_values = 'COARSE' 'FINE' 'MEDIUM' # lexicographical ordering
    descriptors = 'mesh_size'

```

7.3.2.3 solution_level_cost

- [Keywords Area](#)
- [model](#)
- [single](#)
- [solution_level_cost](#)

Cost estimates associated with a set of solution control values.

Specification

Alias: none

Argument(s): REALLIST

Description

Simulation-based models may have an associated `solution_level_control`, which identifies a hierarchy of solution states, such as a set of mesh discretizations from coarse to fine, a set of solver tolerances from loose to tight, etc. In algorithms that manage such a hierarchy and perform optimal resource allocation among the solution states (e.g., multilevel Monte Carlo), it is important to estimate a set of costs associated with each state. These cost estimates can be relative, such as in the example below (lowest cost normalized to 1.)

Note: a scalar solution cost can be specified without an associated solution level control. This is useful when employing a hierarchy of model forms (each model has a scalar solution cost and no solution level control) instead of a hierarchy of discretization levels (one model has a vector-valued solution cost associated with multiple solution levels).

Examples

```

model,
  simulation
    solution_level_control = 'mesh_size'
    solution_level_cost = 1. 8. 64. 512. 4096.

variables,
  uniform_uncertain = 9
  lower_bounds      = 9*-1.
  upper_bounds      = 9* 1.
  discrete_state_set
    integer = 1
      set_values = 4 8 16 32 64
      descriptors = 'mesh_size'

```

7.3.2.4 cost_recovery_metadata

- [Keywords Area](#)
- [model](#)
- [single](#)

- [cost_recovery_metadata](#)

Identify metadata by label for capturing online cost estimates

Specification

Alias: none

Argument(s): STRING

Description

Metadata can be used for a variety of response annotation purposes, including the online recovery of simulation cost (using facilities either implemented in a user's analysis driver or codified in an embedded test driver). This information can then be used by certain algorithms that allocate resources based on relative cost, e.g. multifidelity UQ methods.

Cost estimates may be provided a priori, e.g. using `solution_level_cost`, in cases where they can be assumed fixed for each model form and/or resolution. In cases such as model tuning over a set of hyper-parameters, costs will generally be a function of these hyper-parameters and need to be captured online during initialization of the algorithm (e.g. during evaluation of the pilot sample).

Default Behavior No recovery of cost estimates from the response metadata.

Usage Tips It is common to employ total CPU time, aggregated across all computing cores involved in a (parallel) simulation instance. Wall time, even when the number of cores is fixed, can be subject to additional variability due to system load.

Examples

```
model,
  id_model = 'TUNABLE_LF'
  simulation
  cost_recovery_metadata = 'core_time'

responses,
  response_functions = 1
  descriptors = 'Q'
  no_gradients
  no_hessians
  metadata 'core_time'
```

See Also

These keywords may also be of interest:

- [metadata](#)

7.3.3 surrogate

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

An empirical model that is created from data or the results of a submodel

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		id_surrogates	Identifies the subset of the response functions by number that are to be approximated (the default is all functions).
	Required (<i>Choose One</i>)	Surrogate Category (Group 1)	global	Select a surrogate model with global support
			multipoint	Construct a surrogate from multiple existing training points
			local	Build a locally accurate surrogate from data at a single point
			hierarchical	Hierarchical surrogates use an ordered hierarchy of model fidelities and/or resolutions to trade accuracy versus cost.
			non_hierarchical	Non-hierarchical surrogates employ an unordered set of lower-fidelity models to approximate a truth reference model at reduced cost.

Description

Surrogate models are inexpensive approximate models that are intended to capture the salient features of an expensive high-fidelity model. They can be used to explore the variations in response quantities over regions of the parameter space, or they can serve as inexpensive stand-ins for optimization or uncertainty quantification studies (see, for example, the surrogate-based optimization methods, [surrogate_based_global](#) and [surrogate_based_local](#)).

Surrogate models supported in Dakota are categorized as Data Fitting or Hierarchical, as shown below. Each of these surrogate types provides an approximate representation of a "truth" model which is used to perform the parameter to response mappings. This approximation is built and updated using results from the truth model, called the "training data".

- Data fits:

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Data fitting methods involve construction of an approximation or surrogate model using data (response values, gradients, and Hessians) generated from the original truth model. Data fit methods can be further categorized as local, multipoint, and global approximation techniques, based on the number of points used in generating the data fit.

1. Local: built from response data from a single point in parameter space
 - Taylor series expansion: [taylor_series](#)
Training data consists of a single point, plus gradient and Hessian information.
2. Multipoint: built from two or more points in parameter space, often involving the current and previous iterates of a minimization algorithm.
 - TANA-3: [tana](#)
Training Data comes from a few previously evaluated points
3. Global full space response surface methods:
 - Polynomial regression: [polynomial](#)
 - Gaussian process (Kriging): [gaussian_process](#)
 - Artificial neural network: [neural_network](#)
 - MARS: [mars](#)
 - Radial Basis Functions: [radial_basis](#)
 - Orthogonal polynomials (only supported in PCE/SC for now): [polynomial_chaos](#) and [stoch_collocation](#)

Training data is generated using either a design of experiments method applied to the truth model (specified by [dace_method_pointer](#)), or from saved data (specified by [reuse_points](#)) in a restart database, or an import file.

- Multifidelity/hierarchical:

Multifidelity modeling involves the use of a low-fidelity physics-based model as a surrogate for the original high-fidelity model. The low-fidelity model typically involves a coarser mesh, looser convergence tolerances, reduced element order, or omitted physics.

See [hierarchical](#).

The global and hierarchal surrogates have a correction feature in order to improve the local accuracy of the surrogate models. The correction factors force the surrogate models to match the true function values and possibly true function derivatives at the center point of each trust region. Details can be found on global [correction](#) or hierarchical [correction](#).

Theory

Surrogate models are used extensively in the surrogate-based optimization and least squares methods, in which the goals are to reduce expense by minimizing the number of truth function evaluations and to smooth out noisy data with a global data fit. However, the use of surrogate models is not restricted to optimization techniques; uncertainty quantification and optimization under uncertainty methods are other primary users.

Data Fit Surrogate Models

A surrogate of the {*data fit*} type is a non-physics-based approximation typically involving interpolation or regression of a set of data generated from the original model. Data fit surrogates can be further characterized by the number of data points used in the fit, where a local approximation (e.g., first or second-order Taylor series) uses data from a single point, a multipoint approximation (e.g., two-point exponential approximations (TPEA) or two-point adaptive nonlinearity approximations (TANA)) uses a small number of data points often drawn from the previous iterates of a particular algorithm, and a global approximation (e.g., polynomial response surfaces, kriging/[gaussian_process](#),

neural networks, radial basis functions, splines) uses a set of data points distributed over the domain of interest, often generated using a design of computer experiments.

Dakota contains several types of surface fitting methods that can be used with optimization and uncertainty quantification methods and strategies such as surrogate-based optimization and optimization under uncertainty. These are: polynomial models (linear, quadratic, and cubic), first-order Taylor series expansion, kriging spatial interpolation, artificial neural networks, multivariate adaptive regression splines, radial basis functions, and moving least squares. With the exception of Taylor series methods, all of the above methods listed in the previous sentence are accessed in Dakota through the Surfpack library. All of these surface fitting methods can be applied to problems having an arbitrary number of design parameters. However, surface fitting methods usually are practical only for problems where there are a small number of parameters (e.g., a maximum of somewhere in the range of 30-50 design parameters). The mathematical models created by surface fitting methods have a variety of names in the engineering community. These include surrogate models, meta-models, approximation models, and response surfaces. For this manual, the terms surface fit model and surrogate model are used.

The data fitting methods in Dakota include software developed by Sandia researchers and by various researchers in the academic community.

Multifidelity Surrogate Models

A second type of surrogate is the *{model/hierarchy}* type (also called multifidelity, variable fidelity, variable complexity, etc.). In this case, a model that is still physics-based but is of lower fidelity (e.g., coarser discretization, reduced element order, looser convergence tolerances, omitted physics) is used as the surrogate in place of the high-fidelity model. For example, an inviscid, incompressible Euler CFD model on a coarse discretization could be used as a low-fidelity surrogate for a high-fidelity Navier-Stokes model on a fine discretization.

Surrogate Model Selection

This section offers some guidance on choosing from among the available surrogate model types.

- For Surrogate Based Local Optimization, using the [surrogate_based_local](#) method with a trust region: using the keywords:
 1. [surrogate local taylor_series](#) or
 2. [surrogate multipoint tana](#)
 will probably work best.

If for some reason you wish or need to use a global surrogate (not recommended) then the best of these options is likely to be either:

 1. [surrogate global gaussian_process surfpack](#) or
 2. [surrogate global moving_least_squares](#).
- For Efficient Global Optimization (EGO), the [efficient_global](#) method:

the default surrogate is: [gaussian_process surfpack](#) which is likely to find a more optimal value and/or require fewer true function evaluations than the alternative, [gaussian_process dakota](#). However, the [surfpack](#) will likely take more time to build than the [dakota](#) version. Note that currently the [use_derivatives](#) keyword is not recommended for use with EGO based methods.
- For EGO based global interval estimation, the [global_interval_est ego](#) method:

the default [gaussian_process surfpack](#) will likely work better than the alternative [gaussian_process dakota](#).
- For Efficient Global Reliability Analysis (EGRA), the [global_reliability](#) method:

the [surfpack](#) and [dakota](#) versions of the gaussian process tend to give similar answers with the [dakota](#) version tending to use fewer true function evaluations. Since this is based on EGO, it is likely that the default [surfpack](#) is more accurate, although this has not been rigorously demonstrated.
- For EGO based Dempster-Shafer Theory of Evidence, i.e. the [global_evidence ego](#) method, the default [gaussian_process surfpack](#) often use significantly fewer true function evaluations than the alternative [gaussian_process dakota](#).
- When using a global surrogate to extrapolate, any of the surrogates:

- [gaussian_process surfpack](#)
- [polynomial quadratic](#)
- [polynomial cubic](#)

are recommended.

- When there is over roughly two or three thousand data points and you wish to interpolate (or approximately interpolate) then a Taylor series, Radial Basis Function Network, or Moving Least Squares fit is recommended. The only reason that the [gaussian_process surfpack](#) is not recommended is that it can take a considerable amount of time to construct when the number of data points is very large. Use of the third party MARS package included in Dakota is generally discouraged.
- In other situations that call for a global surrogate, the [gaussian_process surfpack](#) is generally recommended. The `use_derivatives` keyword will only be useful if accurate and inexpensive derivatives are available. Finite difference derivatives are disqualified on both counts. However, derivatives generated by analytical, automatic differentiation, or continuous adjoint techniques can be appropriate. Currently, first order derivatives, i.e. gradients, are the highest order derivatives that can be used to construct the [gaussian_process surfpack](#) model; Hessians will not be used even if they are available.

See Also

These keywords may also be of interest:

- [single](#)
- [nested](#)

7.3.3.1 id_surrogates

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [id_surrogates](#)

Identifies the subset of the response functions by number that are to be approximated (the default is all functions).

Specification

Alias: none

Argument(s): INTEGERLIST

Default: All response functions are approximated

Description

In the `surrogate` model case, the specification first allows a mixture of surrogate and actual response mappings through the use of the optional `id_surrogates` specification. This identifies the subset of the response functions by number that are to be approximated (the default is all functions). The valid response function identifiers range from 1 through the total number of response functions (see [response_functions](#)).

7.3.3.2 global

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)

Select a surrogate model with global support

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Global Surrogate Type (Group 1)	experimental_gaussian_process	Use the Gaussian process regression surrogate from the surrogates module
			gaussian_process	Gaussian Process surrogate model
			mars	Multivariate Adaptive Regression Spline (MARS)
			moving_least_squares	Moving Least Squares surrogate models
			function_train	Global surrogate model based on functional tensor train decomposition
			neural_network	Artificial neural network model
			radial_basis	Radial basis function (RBF) model

			polynomial	Polynomial surrogate model
			experimental_polynomial	Use a deterministic polynomial surrogate
	Optional		domain_decomposition	Piecewise Domain Decomposition for Global Surrogate Models
	Optional (<i>Choose One</i>)	Number of Build Points (Group 2)	total_points	Specified number of training points
			minimum_points	Construct surrogate with minimum number of points
			recommended_points	Construct surrogate with recommended number of points
	Optional (<i>Choose One</i>)	Build Data Source (Group 3)	dace_method_pointer	Specify a method to gather training data
			truth_model_pointer	A surrogate model pointer that guides a method to whether it should use a surrogate model or compute truth function evaluations
	Optional		reuse_points	Surrogate model training data reuse control
	Optional		import_build_points_file	File containing points you wish to use to build a surrogate
	Optional		export_approx_points_file	Output file for surrogate model value evaluations

	Optional	use_derivatives	Use derivative data to construct surrogate models
	Optional	correction	Correction approaches for surrogate models
	Optional	metrics	Compute surrogate quality metrics
	Optional	import_challenge_points_file	Datafile of points to assess surrogate quality

Description

The global surrogate model requires specification of one of the following approximation types:

1. Polynomial
2. Gaussian process (Kriging interpolation)
3. Layered perceptron artificial neural network approximation
4. MARS
5. Moving least squares
6. Radial basis function
7. Voronoi Piecewise Surrogate (VPS)

All these approximations are implemented in SurfPack [36], except for VPS. In addition, a second version of Gaussian process is implemented directly in Dakota.

Training Data

Training data can be taken from prior runs, stored in a datafile, or by running a Design of Experiments method. The keywords listed below are used to determine how to collect training data:

- `dace_method_pointer`
- `reuse_points`
- `import_points_file`
- `use_derivatives` The source of training data is determined by the contents of a provided `import_points_file`, whether `reuse_points` and `use_derivatives` are specified, and the contents of the method block specified by `dace_method_pointer`. `use_derivatives` is a special case, the other keywords are discussed below.

The number of training data points used in building a global approximation is determined by specifying one of three point counts:

1. `minimum_points`: minimum required or minimum "reasonable" amount of training data. Defaults to $d+1$ for d input dimensions for most models, e.g., polynomials override to the number of coefficients required to estimate the requested order.
2. `recommended_points`: recommended number of training data, (this is the default option, if none of the keywords is specified). Defaults to $5*d$, except for polynomials where it's equal to the minimum.

3. `total_points`: specify the number of training data points. However, if the `total_points` value is less than the default `minimum_points` value, the `minimum_points` value is used.

The sources of training data depend on the number of training points, N_{tp} , the number of points in the import file, N_{if} , and the value of `reuse_points`.

- If there is no import file, all training data come from the DACE method
- If there is an import file, all N_{if} points from the file are used, and the remaining $N_{tp} - N_{if}$ points come from the DACE method
- If there is an import file and `reuse_points` is:
 - `none` - all N_{tp} points from DACE method
 - `region` - only the points within a trust region are taken from the import file, and all remaining points are from the DACE method.
 - `all` - (Default) all N_{if} points from the file are used, and the remaining $N_{tp} - N_{if}$ points come from the DACE method

Surrogate Correction

A `correction` model can be added to the constructed surrogate in order to better match the training data. The specified correction method will be applied to the surrogate, and then the corrected surrogate model is used by the method.

Finally, the quality of the surrogate can be tested using the `metrics` and `challenge_points_file` keywords.

Theory

Global methods, also referred to as response surface methods, involve many points spread over the parameter ranges of interest. These surface fitting methods work in conjunction with the sampling methods and design of experiments methods.

Procedures for Surface Fitting

The surface fitting process consists of three steps:

1. selection of a set of design points
2. evaluation of the true response quantities (e.g., from a user-supplied simulation code) at these design points,
3. using the response data to solve for the unknown coefficients (e.g., polynomial coefficients, neural network weights, kriging correlation factors) in the surface fit model.

In cases where there is more than one response quantity (e.g., an objective function plus one or more constraints), then a separate surface is built for each response quantity. Currently, the surface fit models are built using only 0th-order information (function values only), although extensions to using higher-order information (gradients and Hessians) are possible.

Each surface fitting method employs a different numerical method for computing its internal coefficients. For example, the polynomial surface uses a least-squares approach that employs a singular value decomposition to compute the polynomial coefficients, whereas the kriging surface uses Maximum Likelihood Estimation to compute its correlation coefficients. More information on the numerical methods used in the surface fitting codes is provided in the Dakota Developers Manual.

See Also

These keywords may also be of interest:

- [local](#)
- [hierarchical](#)
- [multipoint](#)

7.3.3.2.1 [experimental_gaussian_process](#)

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)

Use the Gaussian process regression surrogate from the surrogates module

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		trend	This keyword enables the use of deterministic polynomial trend function
	Optional		num_restarts	Number of optimization restarts for L-BFGS-B
	Optional (<i>Choose One</i>)	Nugget (Group 1)	nugget	Value for the fixed nugget parameter
			find_nugget	Use regression to estimate the nugget.
	Optional		options_file	Filename for a YAML file that specifies Gaussian process options

	Optional	export_approx_variance_file	Output file for surrogate model variance evaluations
	Optional	export_model	Exports surrogate model in user-specified format(s)
	Optional	import_model	Import surrogate model from archive file

Description

This Gaussian process implementation is contained in Dakota's surrogates module and is considered experimental. It uses gradient-based optimization with restarts to determine hyperparameters and trend coefficients. Nugget and trend estimation are optional.

7.3.3.2.1.1 trend

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [trend](#)

This keyword enables the use of deterministic polynomial trend function

Specification

Alias: none

Argument(s): none

Default: reduced_quadratic

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
			none	No trend function
	Required (<i>Choose One</i>)	Trend Order (Group 1)	constant	Constant trend function
			linear	Use a linear polynomial or trend function

			reduced_quadratic	Quadratic polynomials - main effects only
			quadratic	Use a quadratic polynomial or trend function

Description

Enable the use of a deterministic polynomial trend function whose coefficients are found through regression. The user may choose a constant, linear, or quadratic polynomial.

Default Behavior Defaults to no trend when keyword is absent.

7.3.3.2.1.2 none

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [trend](#)
- [none](#)

No trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.1.3 constant

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [trend](#)
- [constant](#)

Constant trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.1.4 linear

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [trend](#)
- [linear](#)

Use a linear polynomial or trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.1.5 reduced_quadratic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [trend](#)
- [reduced_quadratic](#)

Quadratic polynomials - main effects only

Specification

Alias: none

Argument(s): none

Description

In 2 or more dimensions, this polynomial omits the interaction, or mixed, terms.

7.3.3.2.1.6 quadratic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [trend](#)
- [quadratic](#)

Use a quadratic polynomial or trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.1.7 num_restarts

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [num_restarts](#)

Number of optimization restarts for L-BFGS-B

Specification

Alias: none

Argument(s): INTEGER

Description

Number of optimization runs at random initial guesses for maximum likelihood estimation. Multiple restarts are used to increase the chance of finding the global minimum with a gradient-based (i.e. local) optimization method.

Default Behavior Default value of 20 restarts.

7.3.3.2.1.8 nugget

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [nugget](#)

Value for the fixed nugget parameter

Specification

Alias: none

Argument(s): REAL

Default: None

Description

This keyword is used to specify a value for the nugget parameter. Its presence will turn off nugget estimation.

Default Behavior No nugget when this keyword is absent. Alternatively a nugget may be estimated as part of the regression procedure.

7.3.3.2.1.9 find_nugget

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [find_nugget](#)

Use regression to estimate the nugget.

Specification

Alias: none

Argument(s): INTEGER

Default: None

Description

Estimate a nugget term in the Gaussian process Gram matrix.

Default Behavior Off by default. The user may alternatively specify a fixed nugget.

7.3.3.2.1.10 options_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [options_file](#)

Filename for a YAML file that specifies Gaussian process options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the Gaussian process that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: gp_options.yaml

GP Parameters:
  Sigma Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  scaler name: standardization
  num restarts: 20
  Nugget:
    fixed nugget: 0.0
    estimate nugget: true
  Bounds:
    lower bound: 3.17e-8
    upper bound: 1.0e-2
  gp seed: 42
  standardize response: false
  Trend:
    estimate trend: true
  Options:
    max degree: 2
    reduced basis: false
    p-norm: 1.0
    scaler type: none
    regression solver type: SVD
    verbosity: 1
  kernel type: squared exponential
  Length-scale Bounds:
    lower bound: 1.0e-2
    upper bound: 1.0e2
  verbosity: 1
```

7.3.3.2.1.11 export_approx_variance_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_approx_variance_file](#)

Output file for surrogate model variance evaluations

Specification

Alias: none

Argument(s): STRING

Default: no variance export to a file

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response **variance** values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

See Also

These keywords may also be of interest:

- [export_approx_points_file](#)

7.3.3.2.1.12 custom_annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1      x2      obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9      1.1      0.0002      0.26      0.76
2             0.90009  1.1 0.0001996404857  0.2601620081  0.759955
3             0.89991  1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.3.3.2.1.13 header

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.1.14 eval_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.1.15 interface_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.1.16 annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_approx_variance_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.1.17 freeform

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_approx_variance_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009         1.1 0.0001996404857 0.2601620081 0.759955
0.89991         1.1 0.0002003604863 0.2598380081 0.760045
...

```

7.3.3.2.1.18 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.3.3.2.1.19 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [experimental_gaussian_process](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.1.20 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_model](#)

- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See [export_model](#) and the entries for the format selection keywords for further information.

7.3.3.2.1.21 [text_archive](#)

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.1.22 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.1.23 import_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (<i>Choose One</i>)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the [global](#) surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: a, b, c, d, and inactive state variables s, t, and the surrogate gets built over a, b, c, d. The follow-on study could perform design optimization over active design variables b and d, with inactive state variables a, c, s, t, held at fixed `initial_state` values. In this case the values of a, b, c, d will get mapped to the surrogate and s, t dropped when evaluating it. So it is important the state values for s and t haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    export_model
      filename_prefix = 'gp_export'
      formats = binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

And this model block imports it using a nearly identical model specification:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    import_model
      filename_prefix = 'gp_export'
    binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

7.3.3.2.1.24 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [experimental_gaussian_process](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.1.25 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [import_model](#)

- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.1.26 `binary_archive`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_gaussian_process](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.2 gaussian_process

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)

Gaussian Process surrogate model

Specification

Alias: kriging

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	GP Implementation (Group 1)	dakota	Select the built in Gaussian Process surrogate
			surfpack	Use the Surfpack version of Gaussian Process surrogates
	Optional		export_approx_variance_file	Output file for surrogate model variance evaluations

Description

Use the Gaussian process (GP) surrogate from Surfpack, which is specified using the [surfpack](#) keyword.

An alternate version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.3.3.2.2.1 dakota

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [dakota](#)

Select the built in Gaussian Process surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		point_selection	Enable greedy selection of well-spaced build points
	Optional		trend	Choose a trend function for a Gaussian process surrogate

Description

A second version of GP surrogates was available in prior versions of Dakota. **For now, both versions are supported but the dakota version is deprecated and intended to be removed in a future release.**

Historically these models were drastically different, but in Dakota 5.1, they became quite similar. They now differ in that the Surfpack GP has a richer set of features/options and tends to be more accurate than the Dakota version. Due to how the Surfpack GP handles ill-conditioned correlation matrices (which significantly contributes to its greater accuracy), the [Surfpack](#) GP can be a factor of two or three slower than Dakota's. As of Dakota 5.2, the [Surfpack](#) implementation is the default in all contexts except Bayesian calibration.

More details on the [gaussian_process](#) dakota model can be found in [60].

Dakota's GP deals with ill-conditioning in two ways. First, when it encounters a non-invertible correlation matrix it iteratively increases the size of a "nugget," but in such cases the resulting approximation smooths rather than interpolates the data. Second, it has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

This differs from the `point_selection` option of the Dakota GP which initially chooses a well-spaced subset of points and finds the correlation parameters that are most likely for that one subset.

7.3.3.2.2.2 `point_selection`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [dakota](#)
- [point_selection](#)

Enable greedy selection of well-spaced build points

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Default: no point selection

Description

The Dakota Gaussian Process model has a `point_selection` option (default off) that uses a greedy algorithm to select a well-spaced subset of points prior to the construction of the GP. In this case, the GP will only interpolate the selected subset. Typically, one should not need point selection in trust-region methods because a small number of points are used to develop a surrogate within each trust region. Point selection is most beneficial when constructing with a large number of points, typically more than order one hundred, though this depends on the number of variables and spacing of the sample points.

7.3.3.2.2.3 `trend`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [gaussian_process](#)
- [dakota](#)
- [trend](#)

Choose a trend function for a Gaussian process surrogate

Specification

Alias: none

Argument(s): none

Default: `reduced_quadratic`

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Trend Order (Group 1)	constant	Constant trend function
			linear	Use a linear polynomial or trend function
			reduced_quadratic	Quadratic polynomials - main effects only

Description

The only trend functions that are currently supported are polynomials.

The trend function is selected using the `trend` keyword, with options `constant`, `linear`, or `reduced_quadratic`. The `reduced_quadratic` trend function includes the main effects, but not mixed/interaction terms. The Surpack GP (See [surpack](#)) has the additional option of (a full) `quadratic`.

7.3.3.2.2.4 constant

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [dakota](#)
- [trend](#)
- [constant](#)

Constant trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.2.5 linear

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [dakota](#)
- [trend](#)
- [linear](#)

Use a linear polynomial or trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.2.6 reduced_quadratic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [dakota](#)
- [trend](#)
- [reduced_quadratic](#)

Quadratic polynomials - main effects only

Specification

Alias: none

Argument(s): none

Description

In 2 or more dimensions, this polynomial omits the interaction, or mixed, terms.

7.3.3.2.2.7 surfpack

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)

Use the Surfpack version of Gaussian Process surrogates

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		trend	Choose a trend function for a Gaussian process surrogate
	Optional		optimization_method	Change the optimization method used to compute hyperparameters
	Optional		max_trials	Max number of likelihood function evaluations
	Optional (<i>Choose One</i>)	Nugget (Group 1)	nugget	Specify a nugget to handle ill-conditioning
			find_nugget	Have Surfpack compute a nugget to handle ill-conditioning

	Optional	correlation_lengths	Specify the correlation lengths for the Gaussian process
	Optional	export_model	Exports surrogate model in user-specified format(s)
	Optional	import_model	Import surrogate model from archive file

Description

This keyword specifies the use of the Gaussian process that is incorporated in our surface fitting library called Surfpack.

Several user options are available:

1. Optimization methods:

Maximum Likelihood Estimation (MLE) is used to find the optimal values of the hyper-parameters governing the trend and correlation functions. By default the global optimization method DIRECT is used for MLE, but other options for the optimization method are available. See [optimization_method](#).

The total number of evaluations of the likelihood function can be controlled using the `max_trials` keyword followed by a positive integer. Note that the likelihood function does not require running the "truth" model, and is relatively inexpensive to compute.

2. Trend Function:

The GP models incorporate a parametric trend function whose purpose is to capture large-scale variations. See [trend](#).

3. Correlation Lengths:

Correlation lengths are usually optimized by Surfpack, however, the user can specify the lengths manually. See [correlation_lengths](#).

4. Ill-conditioning

One of the major problems in determining the governing values for a Gaussian process or Kriging model is the fact that the correlation matrix can easily become ill-conditioned when there are too many input points close together. Since the predictions from the Gaussian process model involve inverting the correlation matrix, ill-conditioning can lead to poor predictive capability and should be avoided.

Note that a sufficiently bad sample design could require correlation lengths to be so short that any interpolatory GP model would become inept at extrapolation and interpolation.

The `surfpack` model handles ill-conditioning internally by default, but behavior can be modified using

5. Gradient Enhanced Kriging (GEK).

The `use_derivatives` keyword will cause the Surfpack GP to be constructed from a combination of function value and gradient information (if available).

See notes in the Theory section.

Theory

Gradient Enhanced Kriging

Incorporating gradient information will only be beneficial if accurate and inexpensive derivative information is available, and the derivatives are not infinite or nearly so. Here "inexpensive" means that the cost of evaluating a

function value plus gradient is comparable to the cost of evaluating only the function value, for example gradients computed by analytical, automatic differentiation, or continuous adjoint techniques. It is not cost effective to use derivatives computed by finite differences. In tests, GEK models built from finite difference derivatives were also significantly less accurate than those built from analytical derivatives. Note that GEK's correlation matrix tends to have a significantly worse condition number than Kriging for the same sample design.

This issue was addressed by using a pivoted Cholesky factorization of Kriging's correlation matrix (which is a small sub-matrix within GEK's correlation matrix) to rank points by how much unique information they contain. This reordering is then applied to whole points (the function value at a point immediately followed by gradient information at the same point) in GEK's correlation matrix. A standard non-pivoted Cholesky is then applied to the reordered GEK correlation matrix and a bisection search is used to find the last equation that meets the constraint on the (estimate of) condition number. The cost of performing pivoted Cholesky on Kriging's correlation matrix is usually negligible compared to the cost of the non-pivoted Cholesky factorization of GEK's correlation matrix. In tests, it also resulted in more accurate GEK models than when pivoted Cholesky or whole-point-block pivoted Cholesky was performed on GEK's correlation matrix.

7.3.3.2.2.8 trend

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [trend](#)

Choose a trend function for a Gaussian process surrogate

Specification

Alias: none

Argument(s): none

Default: reduced_quadratic

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Trend Order (Group 1)	constant	Constant trend function
			linear	Use a linear polynomial or trend function
			reduced_quadratic	Quadratic polynomials - main effects only
			quadratic	Use a quadratic polynomial or trend function

Description

The only trend functions that are currently supported are polynomials.

The trend function is selected using the `trend` keyword, with options `constant`, `linear`, or `reduced_quadratic`. The `reduced_quadratic` trend function includes the main effects, but not mixed/interaction terms. The Surfpack GP (See [surfpack](#)) has the additional option of (a full) `quadratic`.

7.3.3.2.2.9 constant

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [trend](#)
- [constant](#)

Constant trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.2.10 linear

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [trend](#)
- [linear](#)

Use a linear polynomial or trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.2.11 reduced_quadratic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [trend](#)
- [reduced_quadratic](#)

Quadratic polynomials - main effects only

Specification

Alias: none

Argument(s): none

Description

In 2 or more dimensions, this polynomial omits the interaction, or mixed, terms.

7.3.3.2.2.12 quadratic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [trend](#)
- [quadratic](#)

Use a quadratic polynomial or trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.2.13 optimization_method

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [optimization_method](#)

Change the optimization method used to compute hyperparameters

Specification

Alias: none

Argument(s): STRING

Default: global

Description

Select the optimization method to compute hyperparameters of the Gaussian Process by specifying one of these arguments:

- `global` (default) - DIRECT method
- `local` - CONMIN method
- `sampling` - generates several random guesses and picks the candidate with greatest likelihood
- `none` - no optimization, pick the center of the feasible region

The `none` option, and the starting location of the `local` optimization, default to the center, in log(correlation length) scale, of the of feasible region.

Surfpack picks a small feasible region of correlation parameters.

Note that we have found the `global` optimization method to be the most robust.

7.3.3.2.2.14 max_trials

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [max_trials](#)

Max number of likelihood function evaluations

Specification

Alias: none

Argument(s): INTEGER

Description

See parent page

7.3.3.2.2.15 nugget

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [nugget](#)

Specify a nugget to handle ill-conditioning

Specification

Alias: none

Argument(s): REAL

Default: None

Description

By default, the Surfpack GP handles ill-conditioning and does not use a nugget. If the user wishes to specify a nugget, there are two approaches.

- The user can specify the value of a nugget with [nugget](#).
- Have Surfpack find the optimal value of the nugget. This is specified by [find_nugget](#). There are two options for `find_nugget`.
 - `find_nugget = 1`: assume that the reciprocal condition number of the correlation matrix R , `rcondR`, is zero and calculate the nugget needed to make the worst case of R not ill-conditioned.
 - `find_nugget = 2`: calculate `rcondR`, which requires a Cholesky factorization. If `rcondR` indicates that R is not ill-conditioned, then kriging uses the Cholesky factorization. Otherwise, if `rcondR` says R is ill conditioned, then kriging will calculate the nugget needed to make the worst case of R not ill conditioned.

`find_nugget = 1` and `2` are similar, the second option just takes more computation (the initial Cholesky factorization) for larger problems.

7.3.3.2.2.16 find_nugget

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [find_nugget](#)

Have Surfpack compute a nugget to handle ill-conditioning

Specification

Alias: none

Argument(s): INTEGER

Default: None

Description

By default, the Surfpack GP handles ill-conditioning and does not use a nugget. If the user wishes to specify a nugget, there are two approaches.

- The user can specify the value of a nugget with [nugget](#).
- Have Surfpack find the optimal value of the nugget. This is specified by [find_nugget](#). There are two options for [find_nugget](#).
 - `find_nugget = 1`: assume that the reciprocal condition number of the correlation matrix R, `rcondR`, is zero and calculate the nugget needed to make the worst case of R not ill-conditioned.
 - `find_nugget = 2`: calculate `rcondR`, which requires a Cholesky factorization. If `rcondR` indicates that R is not ill-conditioned, then kriging uses the Cholesky factorization. Otherwise, if `rcondR` says R is ill conditioned, then kriging will calculate the nugget needed to make the worst case of R not ill conditioned.

`find_nugget = 1` and `2` are similar, the second option just takes more computation (the initial Cholesky factorization) for larger problems.

7.3.3.2.2.17 correlation_lengths

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [correlation_lengths](#)

Specify the correlation lengths for the Gaussian process

Specification

Alias: none

Argument(s): REALLIST

Default: internally computed correlation_lengths

Description

Directly specify `correlation_lengths` as a list of N real numbers where N is the number of input dimensions.

7.3.3.2.2.18 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.3.3.2.2.19 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

7.3.3.2.2.20 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format
	Optional		algebraic_file	Export surrogate model in algebraic format to a file
	Optional		algebraic_console	Export surrogate model in algebraic format to the console

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See [export_model](#) and the entries for the format selection keywords for further information.

7.3.3.2.2.21 [text_archive](#)

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.2.22 `binary_archive`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.2.23 algebraic_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [algebraic_file](#)

Export surrogate model in algebraic format to a file

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern `{prefix}.{response_descriptor}.alg`. See `filename_prefix` for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when `algebraic_console` is specified.

7.3.3.2.2.24 `algebraic_console`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [export_model](#)
- [formats](#)
- [algebraic_console](#)

Export surrogate model in algebraic format to the console

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the `-o` option) in a human-readable "algebraic" format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when `algebraic_file` is specified. Use of `algebraic_file` is preferred over `algebraic_console`, which exists largely to provide a measure of backward compatibility.

7.3.3.2.2.25 `import_model`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)

- [gaussian_process](#)
- [surfpack](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (<i>Choose One</i>)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the `global` surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: `a`, `b`, `c`, `d`, and inactive state variables `s`, `t`, and the surrogate gets built over `a`, `b`, `c`, `d`. The follow-on study could perform design optimization over active design variables `b` and `d`, with inactive state variables `a`, `c`, `s`, `t`, held at fixed `initial_state` values. In this case the values of `a`, `b`, `c`, `d` will get mapped to the surrogate and `s`, `t` dropped

when evaluating it. So it is important the state values for `s` and `t` haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    export_model
      filename_prefix = 'gp_export'
      formats = binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

And this model block imports it using a nearly identical model specification:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    import_model
      filename_prefix = 'gp_export'
      binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

7.3.3.2.26 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.2.27 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [import_model](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.2.28 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [surfpack](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.2.29 export_approx_variance_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [export_approx_variance_file](#)

Output file for surrogate model variance evaluations

Specification

Alias: none

Argument(s): STRING

Default: no variance export to a file

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response **variance** values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

See Also

These keywords may also be of interest:

- [export_approx_points_file](#)

7.3.3.2.2.30 custom_annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1             0.9          1.1          0.0002          0.26          0.76
2             0.90009      1.1 0.0001996404857  0.2601620081  0.759955
3             0.89991      1.1 0.0002003604863  0.2598380081  0.760045
...
```

7.3.3.2.2.31 header

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.2.32 eval_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.2.33 interface_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [export_approx_variance_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.2.34 annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [export_approx_variance_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.2.35 freeform

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [gaussian_process](#)
- [export_approx_variance_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```
      0.9      1.1      0.0002      0.26      0.76
0.90009      1.1 0.0001996404857 0.2601620081 0.759955
0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.3 mars

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)

Multivariate Adaptive Regression Spline (MARS)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_bases	Maximum number of MARS bases
	Optional		interpolation	MARS model interpolation type
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		import_model	Import surrogate model from archive file

Description

This surface fitting method uses multivariate adaptive regression splines from the MARS3.5 package [27] developed at Stanford University.

The MARS reference material does not indicate the minimum number of data points that are needed to create a MARS surface model. However, in practice it has been found that at least $n_{c_{quad}}$, and sometimes as many as 2 to 4 times $n_{c_{quad}}$, data points are needed to keep the MARS software from terminating. Provided that sufficient data samples can be obtained, MARS surface models can be useful in SBO and OUU applications, as well as in the prediction of global trends throughout the parameter space.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Theory

The form of the MARS model is based on the following expression:

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^M a_m B_m(\mathbf{x})$$

where the a_m are the coefficients of the truncated power basis functions B_m , and M is the number of basis functions. The MARS software partitions the parameter space into subregions, and then applies forward and backward regression methods to create a local surface model in each subregion. The result is that each subregion contains its own basis functions and coefficients, and the subregions are joined together to produce a smooth, C^2 -continuous surface model.

MARS is a nonparametric surface fitting method and can represent complex multimodal data trends. The regression component of MARS generates a surface model that is not guaranteed to pass through all of the response data values. Thus, like the quadratic polynomial model, it provides some smoothing of the data.

7.3.3.2.3.1 max_bases

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [mars](#)
- [max_bases](#)

Maximum number of MARS bases

Specification

Alias: none

Argument(s): INTEGER

Description

The maximum number of basis functions allowed in the MARS approximation model.

7.3.3.2.3.2 interpolation

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [interpolation](#)

MARS model interpolation type

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Interpolation Order (Group 1)	linear	Linear interpolation
			cubic	Cubic interpolation

Description

The MARS model interpolation type: linear or cubic.

7.3.3.2.3.3 linear

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [mars](#)
- [interpolation](#)
- [linear](#)

Linear interpolation

Specification

Alias: none

Argument(s): none

Description

Use linear interpolation in the MARS model.

7.3.3.2.3.4 cubic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [interpolation](#)
- [cubic](#)

Cubic interpolation

Specification

Alias: none

Argument(s): none

Description

Use cubic interpolation in the MARS model.

7.3.3.2.3.5 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surpack` executable with an exported model file.

7.3.3.2.3.6 `filename_prefix`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: `exported_surrogate`

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```

surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive

```

7.3.3.2.3.7 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

7.3.3.2.3.8 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.3.9 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [export_model](#)

- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.3.10 import_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (Choose One)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the `global` surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: `a`, `b`, `c`, `d`, and inactive state variables `s`, `t`, and the surrogate gets built over `a`, `b`, `c`, `d`. The follow-on study could perform design optimization over active design variables `b` and `d`, with inactive state variables `a`, `c`, `s`, `t`, held at fixed `initial_state` values. In this case the values of `a`, `b`, `c`, `d` will get mapped to the surrogate and `s`, `t` dropped when evaluating it. So it is important the state values for `s` and `t` haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
```

```

metrics 'root_mean_squared'
  cross_validation folds 5
export_model
  filename_prefix = 'gp_export'
  formats = binary_archive
export_approx_points_file 'gp_values.dat'
export_approx_variance_file 'gp_variance.dat'

```

And this model block imports it using a nearly identical model specification:

```

model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
  import_model
    filename_prefix = 'gp_export'
  binary_archive
  export_approx_points_file 'gp_values.dat'
  export_approx_variance_file 'gp_variance.dat'

```

7.3.3.2.3.11 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

7.3.3.2.3.12 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [import_model](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.3.13 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [mars](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.4 moving_least_squares

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)

Moving Least Squares surrogate models

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		basis_order	Polynomial order for the MLS bases
	Optional		weight_function	Selects the weight function for the MLS model
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		import_model	Import surrogate model from archive file

Description

Moving least squares is a further generalization of weighted least squares where the weighting is "moved" or recalculated for every new point where a prediction is desired [66].

The implementation of moving least squares is still under development. It tends to work well in trust region optimization methods where the surrogate model is constructed in a constrained region over a few points. The present implementation may not work as well globally.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Theory

Moving Least Squares can be considered a more specialized version of linear regression models. In linear regression, one usually attempts to minimize the sum of the squared residuals, where the residual is defined as the difference between the surrogate model and the true model at a fixed number of points.

In weighted least squares, the residual terms are weighted so the determination of the optimal coefficients governing the polynomial regression function, denoted by $\hat{f}(\mathbf{x})$, are obtained by minimizing the weighted sum of squares at N data points:

$$\sum_{n=1}^N w_n (\| \hat{f}(\mathbf{x}_n) - f(\mathbf{x}_n) \|)$$

7.3.3.2.4.1 [basis_order](#)

- [Keywords Area](#)

- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [basis_order](#)

Polynomial order for the MLS bases

Specification

Alias: poly_order

Argument(s): INTEGER

Description

The polynomial order for the moving least squares basis function (default = 2).

7.3.3.2.4.2 weight_function

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [weight_function](#)

Selects the weight function for the MLS model

Specification

Alias: none

Argument(s): INTEGER

Description

The weight function decays as a function of distance from the training data. Specify one of:

- 1 (default): exponential decay in weight function; once differentiable MLS model
- 2: twice differentiable MLS model
- 3: three times differentiable MLS model

7.3.3.2.4.3 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` surpack)
- Artificial neural network (keyword `neural_network`)
- Radial basis Funtions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the surpack executable with an exported model file.

7.3.3.2.4.4 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: `exported_surrogate`

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

7.3.3.2.4.5 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

7.3.3.2.4.6 `text_archive`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surpack executable with an exported model file.

7.3.3.2.4.7 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surpack executable with an exported model file.

7.3.3.2.4.8 import_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)

- [moving_least_squares](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (Choose One)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the [global](#) surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: `a`, `b`, `c`, `d`, and inactive state variables `s`, `t`, and the surrogate gets built over `a`, `b`, `c`, `d`. The follow-on study could perform design optimization over active design variables `b` and `d`, with inactive state variables `a`, `c`, `s`, `t`, held at fixed `initial_state` values. In this case the values of `a`, `b`, `c`, `d` will get mapped to the surrogate and `s`, `t` dropped when evaluating it. So it is important the state values for `s` and `t` haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    export_model
      filename_prefix = 'gp_export'
      formats = binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

And this model block imports it using a nearly identical model specification:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    import_model
      filename_prefix = 'gp_export'
    binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

7.3.3.2.4.9 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.4.10 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [import_model](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.4.11 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [moving_least_squares](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.5 function_train

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)

Global surrogate model based on functional tensor train decomposition

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		regression_type	Type of solver for forming function train approximations by regression
	Optional		max_solver_ - iterations	Maximum iterations in determining polynomial coefficients
	Optional		max_cross_ - iterations	Maximum number of iterations for cross-approximation during a rank adaptation.
	Optional		solver_tolerance	Convergence tolerance for the optimizer used during the regression solve.
	Optional		response_scaling	Perform bounds-scaling on response values prior to surrogate emulation

	Optional	tensor_grid	Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.
	Optional	rounding_tolerance	An accuracy tolerance that is used to guide rounding during rank adaptation.
	Optional	arithmetic_tolerance	A secondary rounding tolerance used for post-processing
	Optional	start_order	(Initial) polynomial order of each univariate function within the functional tensor train.
	Optional	adapt_order	Activate adaptive procedure for determining the best basis order
	Optional	kick_order	increment used when adapting the basis order in function train methods
	Optional	max_order	Maximum polynomial order of each univariate function within the functional tensor train.
	Optional	max_cv_order_candidates	Limit the number of cross-validation candidates for basis order
	Optional	start_rank	The initial rank used for the starting point during a rank adaptation.

	Optional	adapt_rank	Activate adaptive procedure for determining best rank representation
	Optional	kick_rank	The increment in rank employed during each iteration of the rank adaptation.
	Optional	max_rank	Limits the maximum rank that is explored during a rank adaptation.
	Optional	max_cv_rank_candidates	Limit the number of cross-validation candidates for rank

Description

Tensor train decompositions are approximations that exploit low rank structure in an input-output mapping. The form of the approximation can be written as a set of matrix valued products:

$$f_r(x) = F_1(x_1)F_2(x_2)\dots F_d(x_d)$$

Where the "cores" expand to

$$F_k(x_k) = \begin{bmatrix} f_k^{11}(x_k) & \dots & f_k^{1r_k}(x_k) \\ \vdots & \ddots & \vdots \\ f_k^{r_{k-1}1}(x_k) & \dots & f_k^{r_{k-1}r_k}(x_k) \end{bmatrix}$$

An example expansion over four random variables with rank vector (1,7,5,3,1) is

$$f_r(x) = [f_1^{11}(x_1) \quad \dots \quad f_1^{17}(x_1)] \begin{bmatrix} f_2^{11}(x_2) & \dots & f_2^{15}(x_2) \\ \vdots & \ddots & \vdots \\ f_2^{71}(x_2) & \dots & f_2^{75}(x_2) \end{bmatrix} \begin{bmatrix} f_3^{11}(x_3) & \dots & f_3^{13}(x_3) \\ \vdots & \ddots & \vdots \\ f_3^{51}(x_3) & \dots & f_3^{53}(x_3) \end{bmatrix} \begin{bmatrix} f_4^{11}(x_4) \\ \vdots \\ f_4^{31}(x_4) \end{bmatrix}$$

In the current implementation, orthogonal polynomial basis functions (Hermite and Legendre) are employed as the basis functions $f_i^{jk}(x_i)$, although the C3 library will enable additional options in the future.

The number of coefficients that must be computed by the regression solver can be inferred from the construction above. For each QoI, the regression size can be determined as follows:

- For v variables, orders o is a v -vector and ranks r is a $v+1$ -vector
- the first core is a $1 \times r_1$ row vector and contributes $(o_0 + 1)r_1$ terms
- the last core is a $r_{v-1} \times 1$ col vector and contributes $(o_{v-1} + 1)r_{v-1}$ terms
- the middle $v-2$ cores are $r_i \times r_{i+1}$ matrices that contribute $r_i r_{i+1} (o_i + 1)$ terms, $i = 1, \dots, v-2$
- neighboring vec/mat dimensions must match, so there are $v-1$ unique ranks

Usage Tips

This new capability is stabilizing and beginning to be embedded in higher-level strategies such as multilevel-multifidelity algorithms. It is not included in the Dakota build by default, as some C3 library dependencies (CBLAS) can induce small differences in our regression suite.

This capability is also being used as a prototype to explore model-based versus method-based specification of stochastic expansions. While the model specification is stand-alone, it currently requires a corresponding method specification to exercise the model, which can be a generic UQ strategy such as `surrogate_based_uq` method or a `sampling` method. The intent is to migrate function train, polynomial chaos, and stochastic collocation toward model-only specifications that can then be employed in any surrogate/emulator context.

Examples

```
model,  
  id_model = 'FT'  
  surrogate global function_train  
    start_order = 2  
    start_rank = 2  kick_rank = 2  max_rank = 10  
    adapt_rank  
  dace_method_pointer = 'SAMPLING'
```

See Also

These keywords may also be of interest:

- [function_train](#)
- [multilevel_function_train](#)
- [multifidelity_function_train](#)

7.3.3.2.5.1 regression_type

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [regression_type](#)

Type of solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Regression Type (Group 1)	ls	Use least squares solver for forming function train approximations by regression
			rls2	Use regularized regression solver for forming function train approximations

Description

Function train approximations are formed based on regression for the set of coefficients described at [function_train](#). Solver options include least squares and regularized regression, where the latter penalizes high-order terms to mitigate overfitting.

Default Behavior

The default regression solver is least squares.

7.3.3.2.5.2 ls

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [regression_type](#)
- [ls](#)

Use least squares solver for forming function train approximations by regression

Specification

Alias: none

Argument(s): none

Description

Least squares is the standard solver for forming function train approximations based on regression. This option does not include regularization.

7.3.3.2.5.3 rls2

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [function_train](#)
- [regression_type](#)
- [rls2](#)

Use regularized regression solver for forming function train approximations

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		l2_penalty	Penalty value applied in regularized regression solver for function train approximations

Description

RLS2 is a regularized least squares solver that applies an $L2$ penalty to mitigate overfitting. It includes the option to specify a penalty parameter.

See Also

These keywords may also be of interest:

- [l2_penalty](#)

7.3.3.2.5.4 [l2_penalty](#)

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [regression_type](#)
- [rls2](#)
- [l2_penalty](#)

Penalty value applied in regularized regression solver for function train approximations

Specification

Alias: none

Argument(s): REAL

Description

Regularized regression applies a penalty to aspects of the recovered solution (e.g., cardinality, higher-order oscillatory terms, et al.) to mitigate overfitting. This specification provides the scalar penalty parameter (its value is not currently adapted or cross-validated).

Default Behavior

If this penalty parameter is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rls2](#)

7.3.3.2.5.5 max_solver_iterations

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [max_solver_iterations](#)

Maximum iterations in determining polynomial coefficients

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

When using an iterative polynomial coefficient estimation approach, e.g., cross-validation-based solvers, limits the maximum iterations in the coefficient solver.

7.3.3.2.5.6 max_cross_iterations

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)

- [function_train](#)
- [max_cross_iterations](#)

Maximum number of iterations for cross-approximation during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The number of iterations for the cross-approximation algorithm is limited within the `adapt_rank` procedure.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.3.3.2.5.7 solver_tolerance

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [solver_tolerance](#)

Convergence tolerance for the optimizer used during the regression solve.

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-10

Description

Defines the function value and function gradient tolerance for optimizer used solving the regression problem for the functional tensor train coefficients.

7.3.3.2.5.8 response_scaling

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [response_scaling](#)

Perform bounds-scaling on response values prior to surrogate emulation

Specification

Alias: none

Argument(s): none

Default: use original data

Description

Particularly in multifidelity methods based on regression solutions at each level of a model hierarchy, scaling can be an issue since the magnitude of discrepancy data may decay rapidly and regression solvers may employ absolute tolerances in places.

By activating `response_scaling`, the set of response data used for each recovery is scaled to [0,1] based on the minimum value and range of the data set. This prevents loss of accuracy due to imbalances in scale across model levels.

See Also

These keywords may also be of interest:

- [multifidelity_polynomial_chaos](#)
- [multifidelity_function_train](#)

7.3.3.2.5.9 tensor_grid

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [tensor_grid](#)

Use sub-sampled tensor-product quadrature points to build a polynomial chaos expansion.

Specification

Alias: none

Argument(s): none

Default: regression with random sample set

Description

The collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

7.3.3.2.5.10 `rounding_tolerance`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [rounding_tolerance](#)

An accuracy tolerance that is used to guide rounding during rank adaptation.

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

The rounding tolerance is used to measure how closely one can approximate a functional tensor train representation at a step of the optimization procedure using one of lower ranks. If this tolerance is small, then less compression is achieved but greater accuracy is obtained. If this tolerance is big, greater compression is achieved at the expense of accuracy.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [adapt_rank](#)
- [arithmetic_tolerance](#)

7.3.3.2.5.11 arithmetic_tolerance

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [arithmetic_tolerance](#)

A secondary rounding tolerance used for post-processing

Specification

Alias: none

Argument(s): REAL

Default: 1.e-10

Description

After forming one or more function train approximations, this tolerance is used for rolling up final results, in particular, combining multilevel/multifidelity expansions and forming products for higher-order moment estimation. It is generally intended as a looser tolerance than `rounding_tolerance`.

Default Behavior

If this tolerance is not specified, then we adopt the internal C3 default.

See Also

These keywords may also be of interest:

- [rounding_tolerance](#)

7.3.3.2.5.12 start_order

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [start_order](#)

(Initial) polynomial order of each univariate function within the functional tensor train.

Specification**Alias:** order**Argument(s):** INTEGER**Default:** 2**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		dimension_preference	A set of weights specifying the relative importance of each uncertain variable (dimension)

Description

This specifies the initial polynomial order of each univariate function that makes up the functional tensor train decomposition. In the case of `adapt_order`, polynomial order starts from this value and is then advanced; its maximum value is limited by `max_order` and increments are controlled by `kick_order`.

Default Behavior

Default is 2.

See Also

These keywords may also be of interest:

- [adapt_order](#)
- [kick_order](#)
- [max_order](#)

7.3.3.2.5.13 `dimension_preference`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [start_order](#)
- [dimension_preference](#)

A set of weights specifying the relative importance of each uncertain variable (dimension)

Specification

Alias: none

Argument(s): REALLIST

Default: isotropic grids

Description

A set of weights specifying the relative importance of each uncertain variable (dimension). Using this specification leads to anisotropic integrations with differing refinement levels for different random dimensions.

See Also

These keywords may also be of interest:

- [sobol](#)
- [decay](#)

7.3.3.2.5.14 `adapt_order`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [adapt_order](#)

Activate adaptive procedure for determining the best basis order

Specification

Alias: none

Argument(s): none

Default: false

Description

This option activates a cross validation procedure to select the best basis order for representing the current set of data. It searches from `start_order` to `max_order` in increments of `kick_order`.

Default Behavior

No cross validation for basis order.

Examples

This example shows specification of an order adaptation starting at order 2, incrementing by 2, and limited at order 6.

```
model,
  id_model = 'FT'
  surrogate global function_train
  start_rank = 4
  adapt_order start_order = 2 kick_order = 2 max_order = 6
  solver_tolerance = 1e-12
  rounding_tolerance = 1e-12
  dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_order](#)
- [kick_order](#)
- [max_order](#)

7.3.3.2.5.15 kick_order

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [kick_order](#)

increment used when adapting the basis order in function train methods

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

Used in combination with `adapt_order`, `kick_order` defines the size of the increment that is used during the cross validation process, within the range of `start_order` and `max_order`.

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_order](#)

7.3.3.2.5.16 max_order

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [max_order](#)

Maximum polynomial order of each univariate function within the functional tensor train.

Specification

Alias: none

Argument(s): INTEGER

Default: 4

Description

This specification is inactive for curren adaptation options.

In future adaptation options (i.e., cross-approximation adaptation), this will specify the maximum order that can be obtained during adaptation, where polynomial order will start from `start_order` and be limited by `max_order`.

Default Behavior

Max order is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [start_order](#)

7.3.3.2.5.17 max_cv_order_candidates

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [max_cv_order_candidates](#)

Limit the number of cross-validation candidates for basis order

Specification

Alias: none

Argument(s): INTEGER

Default: ushort max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_order](#)
- [increment_max_rank_order](#)
- [polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.3.3.2.5.18 start_rank

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [start_rank](#)

The initial rank used for the starting point during a rank adaptation.

Specification

Alias: rank

Argument(s): INTEGER

Default: 2

Description

The `adapt_rank` procedure increments the rank on each iteration, starting from the rank defined by `start_rank`

Default Behavior

Default is 2.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.3.3.2.5.19 `adapt_rank`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [adapt_rank](#)

Activate adaptive procedure for determining best rank representation

Specification

Alias: none

Argument(s): none

Default: false

Description

The adaptive algorithm proceeds as follows:

1. Start from rank `start_rank` and form an approximation
2. Adapt the current approximation by searching for a solution with lower rank that achieves L2 accuracy within epsilon tolerance of the reference.
3. If a lower rank solution is found with comparable accuracy, then stop. If not, increase the rank by an amount specified by `kick_rank`.
4. Return to step 2 and continue until either `max_rank` is reached or a converged rank (rank less than current reference with comparable accuracy) is found.

Default Behavior

No cross validation for rank.

Examples

This example shows specification of a rank adaptation starting at rank 2, incrementing by 2, and limited at rank 10.

```
model,
  id_model = 'FT'
  surrogate global function_train
  start_order = 5
  adapt_rank start_rank = 2 kick_rank = 2 max_rank = 10
  solver_tolerance = 1e-12
  rounding_tolerance = 1e-12
  dace_method_pointer = 'SAMPLING'
```

Note that `adapt_rank` and `adapt_order` can either be combined or used separately.

See Also

These keywords may also be of interest:

- [start_rank](#)
- [kick_rank](#)
- [max_rank](#)

7.3.3.2.5.20 kick_rank

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [kick_rank](#)

The increment in rank employed during each iteration of the rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 2

Description

The `adapt_rank` procedure increments the rank on each iteration, using an increment defined by `kick_rank`

Default Behavior

Default is 1.

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.3.3.2.5.21 max_rank

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [max_rank](#)

Limits the maximum rank that is explored during a rank adaptation.

Specification

Alias: none

Argument(s): INTEGER

Default: 3

Description

The `adapt_rank` procedure increments the rank on each iteration, and the maximum value is limited by `max_rank`

Default Behavior

Max rank is unbounded (adaptive sweep will stop when CV error increases).

See Also

These keywords may also be of interest:

- [adapt_rank](#)

7.3.3.2.5.22 max_cv_rank_candidates

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [function_train](#)
- [max_cv_rank_candidates](#)

Limit the number of cross-validation candidates for rank

Specification

Alias: none

Argument(s): INTEGER

Default: `size_t` max

Description

When generating refinement candidates by advancing the upper bound used for a cross-validation range, the number of cross-validation can eventually become excessive and hamper algorithm progress. This control limits the maximum number of candidates, which when active, will increase the lower bound for this range as the upper bound is advanced.

See Also

These keywords may also be of interest:

- [increment_max_rank](#)
- [increment_max_rank_order](#)
- [function_train](#)
- [multifidelity_function_train](#)

7.3.3.2.6 neural_network

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)

Artificial neural network model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_nodes	Maximum number of hidden layer nodes
	Optional		range	Range for neural network random weights
	Optional		random_weight	(Inactive) Random weight control
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		import_model	Import surrogate model from archive file

Description

Dakota's artificial neural network surrogate is a stochastic layered perceptron network, with a single hidden layer. Weights for the input layer are chosen randomly, while those in the hidden layer are estimated from data using a variant of the Zimmerman direct training approach [96].

This typically yields lower training cost than traditional neural networks, yet good out-of-sample performance. This is helpful in surrogate-based optimization and optimization under uncertainty, where multiple surrogates may be repeatedly constructed during the optimization process, e.g., a surrogate per response function, and a new surrogate for each optimization iteration.

The neural network is a non parametric surface fitting method. Thus, along with Kriging (Gaussian Process) and MARS, it can be used to model data trends that have slope discontinuities as well as multiple maxima and minima. However, unlike Kriging, the neural network surrogate is not guaranteed to interpolate the data from which it was constructed.

This surrogate can be constructed from fewer than $n_{c_{quad}}$ data points, however, it is a good rule of thumb to use at least $n_{c_{quad}}$ data points when possible.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Theory

The form of the neural network model is

$$\hat{f}(\mathbf{x}) \approx \tanh \{ \mathbf{A}_1 \tanh (\mathbf{A}_0^T \mathbf{x} + \theta_0^T) + \theta_1 \}$$

where \mathbf{x} is the evaluation point in n -dimensional parameter space; the terms \mathbf{A}_0, θ_0 are the random input layer weight matrix and bias vector, respectively; and \mathbf{A}_1, θ_1 are a weight vector and bias scalar, respectively, estimated from training data. These coefficients are analogous to the polynomial coefficients obtained from regression to training data. The neural network uses a cross validation-based orthogonal matching pursuit solver to determine the optimal number of nodes and to solve for the weights and offsets.

7.3.3.2.6.1 max_nodes

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [max_nodes](#)

Maximum number of hidden layer nodes

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: nodes

Argument(s): INTEGER

Default: numTrainingData - 1

Description

Limits the maximum number of hidden layer nodes in the neural network model. The default is to use one less node than the number of available training data points yielding a fully-determined linear least squares problem. However, reducing the number of nodes can help reduce overfitting and more importantly, can drastically reduce surrogate construction time when building from a large data set. (Historically, Dakota limited the number of nodes to 100.)

The keyword `max_nodes` provides an upper bound. Dakota's orthogonal matching pursuit algorithm may further reduce the effective number of nodes in the final model to achieve better generalization to unseen points.

7.3.3.2.6.2 range

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [range](#)

Range for neural network random weights

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): REAL

Description

Controls the range of the input layer random weights in the neural network model. The default range is 2.0, resulting in weights in (-1, 1). These weights are applied after the training inputs have been scaled into [-0.8, 0.8].

7.3.3.2.6.3 random_weight

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [random_weight](#)

(Inactive) Random weight control

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): INTEGER

Description

This option is not currently in use and is likely to be removed

7.3.3.2.6.4 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.3.3.2.6.5 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.6.6 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format
	Optional		algebraic_file	Export surrogate model in algebraic format to a file
	Optional		algebraic_console	Export surrogate model in algebraic format to the console

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

7.3.3.2.6.7 `text_archive`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.6.8 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.6.9 algebraic_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [export_model](#)
- [formats](#)
- [algebraic_file](#)

Export surrogate model in algebraic format to a file

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern `{prefix}.{response_descriptor}.alg`. See `filename_prefix` for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when `algebraic_console` is specified.

7.3.3.2.6.10 algebraic_console

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [export_model](#)
- [formats](#)
- [algebraic_console](#)

Export surrogate model in algebraic format to the console

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the `-o` option) in a human-readable "algebraic" format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when `algebraic_file` is specified. Use of `algebraic_file` is preferred over `algebraic_console`, which exists largely to provide a measure of backward compatibility.

7.3.3.2.6.11 import_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (<i>Choose One</i>)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the [global](#) surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: `a`, `b`, `c`, `d`, and inactive state variables `s`, `t`, and the surrogate gets built over `a`, `b`, `c`, `d`. The follow-on study could perform design optimization over active design variables `b` and `d`, with inactive state variables `a`, `c`, `s`, `t`, held at fixed `initial_state` values. In this case the values of `a`, `b`, `c`, `d` will get mapped to the surrogate and `s`, `t` dropped when evaluating it. So it is important the state values for `s` and `t` haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    export_model
      filename_prefix = 'gp_export'
      formats = binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

And this model block imports it using a nearly identical model specification:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    import_model
      filename_prefix = 'gp_export'
    binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

7.3.3.2.6.12 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

7.3.3.2.6.13 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [import_model](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.6.14 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [neural_network](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.7 radial_basis

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)

Radial basis function (RBF) model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		bases	Initial number of radial basis functions
	Optional		max_pts	Maximum number of RBF CVT points
	Optional		min_partition	(Inactive) Minimum RBF partition
	Optional		max_subsets	Number of trial RBF subsets
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		import_model	Import surrogate model from archive file

Description

Radial basis functions ϕ are functions whose value typically depends on the distance from a center point, called the centroid, \mathbf{c} .

The surrogate model approximation comprises a sum of K weighted radial basis functions:

$$\hat{f}(\mathbf{x}) = \sum_{k=1}^K w_k \phi(\|\mathbf{x} - \mathbf{c}_k\|)$$

These basis functions take many forms, but Gaussian kernels or splines are most common. The Dakota implementation uses a Gaussian radial basis function. The weights are determined via a linear least squares solution approach. See [70] for more details.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

7.3.3.2.7.1 bases

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [bases](#)

Initial number of radial basis functions

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): INTEGER

Description

Initial number of radial basis functions. The default value is the smaller of the number of training points and 100.

7.3.3.2.7.2 max_pts

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [max_pts](#)

Maximum number of RBF CVT points

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): INTEGER

Description

Maximum number of CVT points to use in generating each RBF center. basis computing centroid of each. Defaults to 10 * ([bases](#)). Reducing this will reduce model build time.

7.3.3.2.7.3 min_partition

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [min_partition](#)

(Inactive) Minimum RBF partition

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): INTEGER

Description

This option currently has no effect and will likely be removed.

7.3.3.2.7.4 max_subsets

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [max_subsets](#)

Number of trial RBF subsets

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): INTEGER

Description

Number of passes to take to identify the best subset of basis functions to use. Defaults to the smaller of $3 * (\text{bases})$ and 100.

7.3.3.2.7.5 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.3.3.2.7.6 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.7.7 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format
	Optional		algebraic_file	Export surrogate model in algebraic format to a file
	Optional		algebraic_console	Export surrogate model in algebraic format to the console

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted. See `export_model` and the entries for the format selection keywords for further information.

7.3.3.2.7.8 `text_archive`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.7.9 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.7.10 algebraic_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [export_model](#)
- [formats](#)
- [algebraic_file](#)

Export surrogate model in algebraic format to a file

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern `{prefix}.{response_descriptor}.alg`. See `filename_prefix` for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when `algebraic_console` is specified.

7.3.3.2.7.11 algebraic_console

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [export_model](#)
- [formats](#)
- [algebraic_console](#)

Export surrogate model in algebraic format to the console

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the `-o` option) in a human-readable "algebraic" format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when `algebraic_file` is specified. Use of `algebraic_file` is preferred over `algebraic_console`, which exists largely to provide a measure of backward compatibility.

7.3.3.2.7.12 import_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (<i>Choose One</i>)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the `global` surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: `a`, `b`, `c`, `d`, and inactive state variables `s`, `t`, and the surrogate gets built over `a`, `b`, `c`, `d`. The follow-on study could perform design optimization over active design variables `b` and `d`, with inactive state variables `a`, `c`, `s`, `t`, held at fixed `initial_state` values. In this case the values of `a`, `b`, `c`, `d` will get mapped to the surrogate and `s`, `t` dropped when evaluating it. So it is important the state values for `s` and `t` haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    export_model
      filename_prefix = 'gp_export'
      formats = binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

And this model block imports it using a nearly identical model specification:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    import_model
      filename_prefix = 'gp_export'
    binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

7.3.3.2.7.13 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

7.3.3.2.7.14 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [import_model](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.7.15 `binary_archive`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [radial_basis](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.8 polynomial

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)

Polynomial surrogate model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Polynomial Order (Group 1)	basis_order	Polynomial order
			linear	Use a linear polynomial or trend function
			quadratic	Use a quadratic polynomial or trend function
			cubic	Use a cubic polynomial
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		import_model	Import surrogate model from archive file

Description

Linear, quadratic, and cubic polynomial surrogate models are available in Dakota. The utility of the simple polynomial models stems from two sources:

- over a small portion of the parameter space, a low-order polynomial model is often an accurate approximation to the true data trends
- the least-squares procedure provides a surface fit that smooths out noise in the data.

Local surrogate-based optimization methods ([surrogate_based_local](#)) are often successful when using polynomial models, particularly quadratic models. However, a polynomial surface fit may not be the best choice for modeling data trends globally over the entire parameter space, unless it is known a priori that the true data trends are close to linear, quadratic, or cubic. See [\[65\]](#) for more information on polynomial models.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Theory

The form of the linear polynomial model is

$$\hat{f}(\mathbf{x}) \approx c_0 + \sum_{i=1}^n c_i x_i$$

the form of the quadratic polynomial model is:

$$\hat{f}(\mathbf{x}) \approx c_0 + \sum_{i=1}^n c_i x_i + \sum_{i=1}^n \sum_{j \geq i}^n c_{ij} x_i x_j$$

and the form of the cubic polynomial model is:

$$\hat{f}(\mathbf{x}) \approx c_0 + \sum_{i=1}^n c_i x_i + \sum_{i=1}^n \sum_{j \geq i}^n c_{ij} x_i x_j + \sum_{i=1}^n \sum_{j \geq i}^n \sum_{k \geq j}^n c_{ijk} x_i x_j x_k$$

In all of the polynomial models, $\hat{f}(\mathbf{x})$ is the response of the polynomial model; the x_i, x_j, x_k terms are the components of the n -dimensional design parameter values; the $c_0, c_i, c_{ij}, c_{ijk}$ terms are the polynomial coefficients, and n is the number of design parameters. The number of coefficients, n_c , depends on the order of polynomial model and the number of design parameters. For the linear polynomial:

$$n_{c_{linear}} = n + 1$$

for the quadratic polynomial:

$$n_{c_{quad}} = \frac{(n+1)(n+2)}{2}$$

and for the cubic polynomial:

$$n_{c_{cubic}} = \frac{(n^3 + 6n^2 + 11n + 6)}{6}$$

There must be at least n_c data samples in order to form a fully determined linear system and solve for the polynomial coefficients. In Dakota, a least-squares approach involving a singular value decomposition numerical method is applied to solve the linear system.

7.3.3.2.8.1 basis_order

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [basis_order](#)

Polynomial order

Specification

Alias: none

Argument(s): INTEGER

Description

The polynomial order for the polynomial regression model (default = 2).

7.3.3.2.8.2 linear

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [linear](#)

Use a linear polynomial or trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.8.3 quadratic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [quadratic](#)

Use a quadratic polynomial or trend function

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.8.4 cubic

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [cubic](#)

Use a cubic polynomial

Specification

Alias: none

Argument(s): none

Description

See parent page

7.3.3.2.8.5 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)
- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surfpack` executable with an exported model file.

7.3.3.2.8.6 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [polynomial](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.8.7 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [export_model](#)

- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format
	Optional		algebraic_file	Export surrogate model in algebraic format to a file
	Optional		algebraic_console	Export surrogate model in algebraic format to the console

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See [export_model](#) and the entries for the format selection keywords for further information.

7.3.3.2.8.8 [text_archive](#)

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.8.9 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [export_model](#)
- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.8.10 algebraic_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [export_model](#)
- [formats](#)
- [algebraic_file](#)

Export surrogate model in algebraic format to a file

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to a file in a human-readable "algebraic" format. The file is named using the pattern `{prefix}.{response_descriptor}.alg`. See `filename_prefix` for further information about exported surrogate file naming. The file contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the file matches exactly the output written to the console when `algebraic_console` is specified.

7.3.3.2.8.11 algebraic_console

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [export_model](#)
- [formats](#)
- [algebraic_console](#)

Export surrogate model in algebraic format to the console

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

After the surrogate model has been built, Dakota will export it to the console (screen, or output file if Dakota was run using the `-o` option) in a human-readable "algebraic" format. The output contains sufficient information for the user to (re)construct and evaluate the model outside of Dakota.

Expected Output

The format depends on the type of surrogate model, but in general will include a LaTeX-like representation of the analytic form of the model to aid tool development, all needed model hyperparameters, and headers describing the shape or dimension of the provided data.

The output written to the screen for the exported model matches exactly the output written to file when `algebraic_file` is specified. Use of `algebraic_file` is preferred over `algebraic_console`, which exists largely to provide a measure of backward compatibility.

7.3.3.2.8.12 import_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (<i>Choose One</i>)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the `global` surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: `a`, `b`, `c`, `d`, and inactive state variables `s`, `t`, and the surrogate gets built over `a`, `b`, `c`, `d`. The follow-on study could perform design optimization over active design variables `b` and `d`, with inactive state variables `a`, `c`, `s`, `t`, held at fixed `initial_state` values. In this case the values of `a`, `b`, `c`, `d` will get mapped to the surrogate and `s`, `t` dropped when evaluating it. So it is important the state values for `s` and `t` haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    export_model
      filename_prefix = 'gp_export'
      formats = binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

And this model block imports it using a nearly identical model specification:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
      metrics 'root_mean_squared'
      cross_validation folds 5
    import_model
      filename_prefix = 'gp_export'
    binary_archive
      export_approx_points_file 'gp_values.dat'
      export_approx_variance_file 'gp_variance.dat'
```

7.3.3.2.8.13 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

7.3.3.2.8.14 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [import_model](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.8.15 `binary_archive`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [polynomial](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.9 experimental_polynomial

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)

Use a deterministic polynomial surrogate

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		basis_order	Total degree of the polynomial basis
	Optional		options_file	Filename for a YAML file that specifies polynomial surrogate options
	Optional		export_model	Exports surrogate model in user-specified format(s)
	Optional		import_model	Import surrogate model from archive file

Description

Compute a polynomial surrogate for a specified total order by least-squares regression. This model is implemented in Dakota's surrogates module and is considered experimental.

7.3.3.2.9.1 basis_order

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [basis_order](#)

Total degree of the polynomial basis

Specification

Alias: none

Argument(s): INTEGER

Description

This keyword's argument specifies the terms in the polynomial according to a total order scheme.

7.3.3.2.9.2 options_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [options_file](#)

Filename for a YAML file that specifies polynomial surrogate options

Specification

Alias: none

Argument(s): STRING

Default: no advanced options file

Description

The YAML file contains configuration options that are used to populate a Teuchos ParameterList used by the polynomial that will override other keyword-specified parameters. Missing options in the YAML file are set to default values.

Examples

```
# FILE: polynomial_options.yaml

Polynomial Parameters:
  max degree: 3
  scaler type: none
  reduced basis: false
  p-norm: 1.0
  regression solver type: SVD
  standardize response: false
  verbosity: 1
```

7.3.3.2.9.3 export_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [experimental_polynomial](#)
- [export_model](#)

Exports surrogate model in user-specified format(s)

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required		formats	Formats for surrogate model export

Description

Export the surrogate for later re-import to Dakota or evaluation using the `surfpack` executable (`bin/surfpack`) or a user-developed tool. Export format is controlled using the `formats` specification. Four formats are available in Dakota; however, not all have been enabled for all surrogates.

The four formats are:

- `text_archive` - Plain-text, machine-readable archive for re-import or use with the `surfpack` executable
- `binary_archive` - Binary, machine-readable archive for re-import or use with the `surfpack` executable
- `algebraic_file` - Plain-text, human-readable file intended for use with user-created tools; not compatible with Dakota or the `surfpack` executable
- `algebraic_console` - Print the model in algebraic format to the screen; not compatible with Dakota or the `surfpack` executable

These global surrogates can be exported in all four formats:

- Gaussian process (keyword `gaussian_process` `surfpack`)
- Artificial neural network (keyword `neural_network`)
- Radial basis Functions (keyword `radial_basis`)

- Polynomial (keyword `polynomial`)

However, for experimental Gaussian Process and polynomial models as well as Multivariate Adaptive Regression Spline (keyword `mars`) and moving least squares (keyword `moving_least_squares`) models, only `text_archive` and `binary_archive` formats may be used.

Currently, no other surrogate models can be exported.

Default Behavior

No export.

Expected Output

Output depends on selected format; see the `formats` specification.

Additional Discussion

The Dakota examples include a demonstration of using the `surpack` executable with an exported model file.

7.3.3.2.9.4 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [export_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: `exported_surrogate`

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
  export_model
    filename_prefix = 'my_surrogate'
    formats
      algebraic_file
      binary_archive
```

7.3.3.2.9.5 formats

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [export_model](#)
- [formats](#)

Formats for surrogate model export

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		text_archive	Surrogate model plain-text archive file format
	Optional		binary_archive	Surrogate model binary archive file format

Description

Select from among the 2-4 available export formats available for this surrogate. Multiple selections are permitted.

See `export_model` and the entries for the format selection keywords for further information.

7.3.3.2.9.6 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [export_model](#)
- [formats](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.9.7 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [export_model](#)

- [formats](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.9.8 import_model

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [import_model](#)

Import surrogate model from archive file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		filename_prefix	User-customizable portion of exported/imported surrogate model filenames
	Required (Choose One)	Surrogate Import Format (Group 1)	text_archive	Surrogate model plain-text archive file format
			binary_archive	Surrogate model binary archive file format

Description

Imports a surrogate model from a file in `binary_archive` or `text_archive` format, typically instead of constructing it from generated or imported data. Importing can result in significant time savings with some models, such as Gaussian processes. The file from which to import is further specified with the child keywords `filename_prefix` and `binary_archive` or `text_archive`.

Default Behavior

When used in the context of an iteratively adapted or rebuilt surrogate, the imported model will only replace the initial surrogate model. Subsequent surrogate model builds require imported build points or a design of experiments method specified through `dace_method_pointer`.

Usage Tips

When importing a surrogate model, *it is crucial* that the `global` surrogate model part of the Dakota input file be identical for export and import, except for changing `export_model` and its child keywords to those needed for `import_model`. Any other keywords such as specifying a `dace_iterator` or imported points must remain intact to satisfy internal surrogate constructor requirements.

All variables over which the surrogate model was built (typically the active variables) when exported must be present in the Dakota study that imports it. The variables in the importing study will be matched by descriptor to those in the surrogate model. This allows the order and type of variables to change between export and import, for example to fix unimportant parameters as state variables even if they were previously active design variables.

For example, suppose the original Dakota study for the surrogate build had active uniform uncertain variables: `a`, `b`, `c`, `d`, and inactive state variables `s`, `t`, and the surrogate gets built over `a`, `b`, `c`, `d`. The follow-on study could perform design optimization over active design variables `b` and `d`, with inactive state variables `a`, `c`, `s`, `t`, held at fixed `initial_state` values. In this case the values of `a`, `b`, `c`, `d` will get mapped to the surrogate and `s`, `t` dropped when evaluating it. So it is important the state values for `s` and `t` haven't changed in some way that invalidates the surrogate.

While it might be possible to import surrogates externally-generated using the `surfpack` binary or the experimental `dakota.surrogates` Python module, it is untested and not the primary use case for this capability.

Examples

The following model block exports a Gaussian process after it's built:

```
model
  id_model = 'SURR_M'
  surrogate global
    dace_method_pointer = 'DACE'
    experimental_gaussian_process
```

```

metrics 'root_mean_squared'
  cross_validation folds 5
export_model
  filename_prefix = 'gp_export'
  formats = binary_archive
export_approx_points_file 'gp_values.dat'
export_approx_variance_file 'gp_variance.dat'

```

And this model block imports it using a nearly identical model specification:

```

model
  id_model = 'SURR_M'
  surrogate global
  dace_method_pointer = 'DACE'
  experimental_gaussian_process
  metrics 'root_mean_squared'
  cross_validation folds 5
  import_model
  filename_prefix = 'gp_export'
  binary_archive
  export_approx_points_file 'gp_values.dat'
  export_approx_variance_file 'gp_variance.dat'

```

7.3.3.2.9.9 filename_prefix

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [import_model](#)
- [filename_prefix](#)

User-customizable portion of exported/imported surrogate model filenames

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): STRING

Default: exported_surrogate

Description

Dakota surrogate model files are exported and imported using one file per response, per requested format. The files are named using the pattern `{prefix}.{response_descriptor}.{extension}`. This `filename_prefix` keyword is used to supply the prefix portion of the pattern.

The `response_descriptor` portion of the pattern is filled in using the response [descriptors](#) provided by the user (or, if none are specified, descriptors automatically generated by Dakota). Extension is a three or four letter string that depends on the format.

Examples

This input snippet directs Dakota to write one algebraic format file and one binary archive file for each response. The names of the files will follow the patterns `my_surrogate.{response_descriptor}.alg` (for the algebraic files) and `my_surrogate.{response_descriptor}.bsps` (for the binary files).

```
surrogate global gaussian_process surfpack
export_model
  filename_prefix = 'my_surrogate'
  formats
    algebraic_file
    binary_archive
```

7.3.3.2.9.10 text_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [import_model](#)
- [text_archive](#)

Surrogate model plain-text archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a plain-text archive format.

Experimental surrogates will export to (import from) text archives named `{prefix}.{response_descriptor}.txt`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.sps`, in which 'sps' stands for Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.9.11 binary_archive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [experimental_polynomial](#)
- [import_model](#)
- [binary_archive](#)

Surrogate model binary archive file format

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Description

When specified, the surrogate model will be exported (or imported) using a binary archive format.

Experimental surrogates will export to (import from) binary archives named `{prefix}.{response_descriptor}.bin`.

Other (Surfpack) surrogates will export/import using files named `{prefix}.{response_descriptor}.bsps`, in which 'bsps' stands for binary Surfpack surrogate.

See `filename_prefix` for further information about surrogate file naming.

Examples

The Dakota examples include a demonstration of using the surfpack executable with an exported model file.

7.3.3.2.10 domain_decomposition

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [domain_decomposition](#)

Piecewise Domain Decomposition for Global Surrogate Models

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		cell_type	Type of the Geometric Cells Used for the Piecewise Decomposition Option of Global Surrogates
	Optional		support_layers	Optional Number of Support Layers for the Piecewise Decomposition Option of Global Surrogates
	Optional		discontinuity_-detection	Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

Description

Typical regression techniques use all available sample points to build continuous approximations the underlying function.

An alternative option is to use piecewise decomposition to locally approximate the function at some point using a few sample points from its neighborhood only. This option currently supports Polynomial Regression, Gaussian Process (GP) Interpolation, and Radial Basis Functions (RBF) Regression. It requires a decomposition cell type (currently set to be Voronoi cells). Optional parameters are: the number of layers of neighbors used to solve the regression problem (default is one layer), and an optional discontinuity detection capability (identified by a user-input jump or gradient threshold).

The method can also make use of the gradient and Hessian information, if available. The user needs to specify the keyword `user_derivatives`.

7.3.3.2.10.1 `cell_type`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [domain_decomposition](#)

- [cell_type](#)

Type of the Geometric Cells Used for the Piecewise Decomposition Option of Global Surrogates

Specification

Alias: none

Argument(s): STRING

Description

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

This option requires a decomposition cell type that can vary from structured grid boxes, to polygonal Voronoi cells. Currently, this option only supports Voronoi cells.

7.3.3.2.10.2 support_layers

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [domain_decomposition](#)
- [support_layers](#)

Optional Number of Support Layers for the Piecewise Decomposition Option of Global Surrogates

Specification

Alias: none

Argument(s): INTEGER

Description

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The neighborhood of a cell is parameterized via a number of (support layers). The default value is set to one layer of neighbors (cells that share direct edges with the cell under study). One more support layer would include the neighbors of that cell's neighbors, and so on.

7.3.3.2.10.3 discontinuity_detection

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [domain_decomposition](#)

- [discontinuity_detection](#)

Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Threshold Mode (Group 1)	jump_threshold	Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates
			gradient_threshold	Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

Description

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The domain decomposition algorithm supports an optional discontinuity detection capability where seeds across a user-input discontinuity threshold are not considered neighbors when building the approximate connectivity Delaunay graph. Alternatively, the domain is split into patches that trap discontinuities between them. This capability is specified by either jump or gradient threshold values in the input spec.

7.3.3.2.10.4 [jump_threshold](#)

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [domain_decomposition](#)
- [discontinuity_detection](#)

- [jump_threshold](#)

Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

Specification

Alias: none

Argument(s): REAL

Description

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The domain decomposition algorithm supports an optional discontinuity detection capability where seeds across a user-input discontinuity threshold are not considered neighbors when building the approximate connectivity Delaunay graph. Alternatively, the domain is split into patches that trap discontinuities between them. This capability can be specified using a jump threshold value in the input spec.

7.3.3.2.10.5 [gradient_threshold](#)

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [domain_decomposition](#)
- [discontinuity_detection](#)
- [gradient_threshold](#)

Gradient Threshold Parameter of the Optional Discontinuity Detection Capability for the Piecewise Decomposition Option of Global Surrogates

Specification

Alias: none

Argument(s): REAL

Description

The piecewise decomposition option for global surrogates is used to locally approximate a function at some point using a few sample points from its neighborhood.

The domain decomposition algorithm supports an optional discontinuity detection capability where seeds across a user-input discontinuity threshold are not considered neighbors when building the approximate connectivity Delaunay graph. Alternatively, the domain is split into patches that trap discontinuities between them. This capability can be specified using a gradient threshold value in the input spec.

7.3.3.2.11 total_points

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [total_points](#)

Specified number of training points

Specification

Alias: none

Argument(s): INTEGER

Default: recommended_points

Description

See parent page.

7.3.3.2.12 minimum_points

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [minimum_points](#)

Construct surrogate with minimum number of points

Specification

Alias: none

Argument(s): none

Description

The minimum is $d+1$, for d input dimensions, except for polynomials. See parent page.

7.3.3.2.13 recommended_points

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [recommended_points](#)

Construct surrogate with recommended number of points

Specification

Alias: none

Argument(s): none

Description

This is the default option. It requires $5 \times d$ build points for d input dimensions, except for polynomial models. See parent page.

7.3.3.2.14 dace_method_pointer

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [dace_method_pointer](#)

Specify a method to gather training data

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: no design of experiments data

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		auto_refinement	Experimental auto-refinement of surrogate model

Description

The number of training points and the sources are specified on [global](#), as well as the number of new training points required.

New training points are gathered by running the "truth" model using the method specified by `dace_method_pointer`. The DACE method will only be invoked if it has new samples to perform, and if new samples are required and no DACE iterator has been provided, an error will result.

The `dace_method_pointer` points to design of experiments method block used to generate truth model data.

Permissible methods include: Monte Carlo (random) sampling, Latin hypercube sampling, orthogonal array sampling, central composite design sampling, and Box-Behnken sampling.

Note that the number of samples specified in the method block may be overwritten, if the requested number of samples is less than [minimum_points](#).

7.3.3.2.14.1 auto_refinement

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [dace_method_pointer](#)
- [auto_refinement](#)

Experimental auto-refinement of surrogate model

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Default: no refinement

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		max_iterations	Number of iterations allowed for optimizers and adaptive UQ methods
	Optional		max_function_evaluations	Number of function evaluations allowed for optimizers
	Optional		convergence_tolerance	Cross-validation threshold for surrogate convergence

	Optional	soft_convergence_limit	Maximum number of iterations without improvement in cross-validation
	Optional	cross_validation_metric	Choice of error metric to satisfy

Description

(Experimental option) Automatically refine the surrogate model until desired cross-validation quality is achieved. Refinement is accomplished by iteratively adding more data to the training set until the cross-validation `convergence_tolerance` is achieved, or `max_function_evaluations` or `max_iterations` is exceeded.

The amount of new training data that is incorporated each iteration is specified in the DACE method that is referred to by the model's `dace_method_pointer`. See [refinement_samples](#) for more information.

7.3.3.2.14.2 max_iterations

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [dace_method_pointer](#)
- [auto_refinement](#)
- [max_iterations](#)

Number of iterations allowed for optimizers and adaptive UQ methods

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 100

Description

The maximum number of iterations is used as a stopping criterion for optimizers and some adaptive UQ methods. If it has not reached any other stopping criteria first, the method will stop after it has performed `max_iterations` iterations. In cases where there are multiple nested iterations to be controlled, `max_iterations` generally controls the outer-most context.

Default Behavior

Default value is 100.

See Also

These keywords may also be of interest:

- [max_function_evaluations](#)

7.3.3.2.14.3 max_function_evaluations

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [dace_method_pointer](#)
- [auto_refinement](#)
- [max_function_evaluations](#)

Number of function evaluations allowed for optimizers

Topics

This keyword is related to the topics:

- [method_independent_controls](#)

Specification

Alias: none

Argument(s): INTEGER

Default: 1000

Description

The maximum number of function evaluations is used as a stopping criterion for optimizers. If it has not reached any other stopping criteria first, the optimizer will stop after it has performed `max_function_evaluations` evaluations. See also `max_iterations`.

Some optimizers (e.g. `ncsu_direct`) may run past this limit in the course of an iteration step that began before `max_function_evaluations` was exceeded.

Default Behavior

Default value is 1000.

7.3.3.2.14.4 convergence_tolerance

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [dace_method_pointer](#)

- [auto_refinement](#)
- [convergence_tolerance](#)

Cross-validation threshold for surrogate convergence

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-4

Description

The surrogate model will be refined until the selected `cross_validation_metric` falls below this convergence tolerance.

Default Behavior The default is 1e-4.

7.3.3.2.14.5 `soft_convergence_limit`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [dace_method_pointer](#)
- [auto_refinement](#)
- [soft_convergence_limit](#)

Maximum number of iterations without improvement in cross-validation

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

The purpose of this control is to cease auto-refinement when improvement in the cross-validation score appears to have stalled or occurs too slowly.

A rolling average of improvement in the cross-validation score is computed over `soft_convergence_limit` iterations. Auto-refinement is halted if the average becomes smaller than the `convergence_tolerance`. The average is computed but ignored until the number of iterations has exceeded `soft_convergence_limit`.

The default setting of 0 disables soft convergence.

7.3.3.2.14.6 cross_validation_metric

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [dace_method_pointer](#)
- [auto_refinement](#)
- [cross_validation_metric](#)

Choice of error metric to satisfy

Specification

Alias: none

Argument(s): STRING

Default: root-mean-squared error

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		folds	Number of cross validation folds

Description

The cross-validation score that the auto-refinement routine seeks to satisfy is based on an error metric. Any of the surrogate diagnostic [metrics](#) may be used. These include:

- root_mean_squared
- mean_abs
- rsquared
- sum_squared
- mean_squared
- sum_abs
- max_abs

Default Behavior The default metric is the the root mean squared error ('root_mean_squared').

7.3.3.2.14.7 folds

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)

- [dace_method_pointer](#)
- [auto_refinement](#)
- [cross_validation_metric](#)
- [folds](#)

Number of cross validation folds

Specification

Alias: none

Argument(s): INTEGER

Default: 10

Description

Number of folds (partitions) of the training data to use in cross validation (default 10).

7.3.3.2.15 `truth_model_pointer`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [truth_model_pointer](#)

A surrogate model pointer that guides a method to whether it should use a surrogate model or compute truth function evaluations

Specification

Alias: `actual_model_pointer`

Argument(s): STRING

Description

Dakota methods use global surrogate models to compute surrogate function approximations. They also need to know the true function evaluations. A global surrogate model now must have a `truth_model_pointer` keyword to decide for the method whether to evaluate the global surrogate model, or compute the true function evaluations if `truth_model_pointer = 'TRUTH'`.

7.3.3.2.16 `reuse_points`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [reuse_points](#)

Surrogate model training data reuse control

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: reuse_samples

Argument(s): none

Default: all for import; none otherwise

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Reuse Domain (Group 1)	all	Option for reuse_points
			region	Option for reuse_points
			none	Option for reuse_points

Description

Dakota's global surrogate methods rely on training data, which can either come from evaluation of a "truth" model, which is generated by the method specified with [dace_method_pointer](#), from a file of existing training data, identified by [import_build_points_file](#), or both.

The `reuse_points` keyword controls the amount of training data used in building a surrogate model, either initially, or during iterative rebuild, as in surrogate-based optimization. If [import_build_points_file](#) is specified, `reuse_points` controls how the file contents are used. If used during iterative rebuild, it controls what data from previous surrogate builds is reused in building the current model.

- `all` (default for file import) - use all points in the file or available from previous builds
- `region` - use only the points falling in the current trust region (see [surrogate_based_local](#))
- `none` (default when no import) - ignore the contents of the file or previous build points, and gather new training data using the specified DACE method

7.3.3.2.16.1 all

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [reuse_points](#)
- [all](#)

Option for reuse_points

Specification

Alias: none

Argument(s): none

Description

This is described on the parent page.

7.3.3.2.16.2 region

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [reuse_points](#)
- [region](#)

Option for `reuse_points`

Specification

Alias: none

Argument(s): none

Description

This is described on the parent page.

7.3.3.2.16.3 none

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [reuse_points](#)
- [none](#)

Option for `reuse_points`

Specification

Alias: none

Argument(s): none

Description

This is described on the parent page.

7.3.3.2.17 import_build_points_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)

File containing points you wish to use to build a surrogate

Specification

Alias: import_points_file samples_file

Argument(s): STRING

Default: no point import from a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

The `import_build_points_file` allows the user to specify a file that contains a list of points and truth model responses used to construct a surrogate model. These can be used by all methods that (explicitly, e.g. surrogate-based optimization, or implicitly, e.g. efficient global optimization) operate on a surrogate. In particular, these points and responses are used in place of truth model evaluations to construct the initial surrogate. When used to construct surrogate models or emulators these are often called build points or training data.

Default Behavior

By default, methods do not import points from a file.

Usage Tips

Dakota parses input files without regard to whitespace, but the `import_build_points_file` must be in one of three formats:

- `annotated` (default)

- `custom_annotated`
- `freeform`

Examples

```
method
  polynomial_chaos
    expansion_order = 4
    import_build_points_file = 'dakota_uq_rosenbrock_pce_import.annot.pts.dat'
```

7.3.3.2.17.1 custom_annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether header row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.17.2 header

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		use_variable_labels	Validate/use variable labels from tabular file header

Description

See description of parent `custom_annotated`

7.3.3.2.17.3 `use_variable_labels`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [header](#)
- [use_variable_labels](#)

Validate/use variable labels from tabular file header

Specification

Alias: none

Argument(s): none

Description

When importing global surrogate training data (or challenge evaluation points) from a tabular data file containing a header (`annotated` or `custom_annotated` header), this keyword toggles validation of variable labels present in the header row.

The labels are validated against the descriptors of the variables being imported to. If the tabular file labels can be rearranged to match the expected labels, the columns in the data file will be reordered on read to match the Dakota variable order. If the read labels are not a permutation of, nor equal to, the expected labels, an error will result.

Default Behavior

When not specified, variable labels will be read, but not strictly enforced or reordered. A warning will be issued if the variable labels are not as expected, and guidance offered if they can be permuted to match expectations.

Expected Output

Console output will be generated for any warnings, as well as to indicate whether any variable reordering is taking place.

Usage Tips

The use of this keyword is recommended when importing header-annotated tabular data files where the variables are appropriately labeled. Tabular files do not always contain response labels, so no attempt is made to disambiguate variable from response labels. The variable labels must appear contiguously in the header after any leading column IDs such as `'eval_id'` or `'interface'`.

Examples

This example enforces variable labels for both build and challenge points

```
model
  id_model = 'SURR'
  surrogate global
    polynomial quadratic
  import_build_points_file = 'dakota_surrogate_import.unc_fixedothers.dat'
    annotated use_variable_labels
  challenge_points_file = 'dakota_surrogate_import.dat'
    annotated use_variable_labels
```

7.3.3.2.17.4 eval_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.17.5 interface_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.17.6 annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		use_variable_labels	Validate/use variable labels from tabular file header

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated` header `eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9          1.1          0.0002          0.26          0.76
2          NO_ID            0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.17.7 use_variable_labels

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [annotated](#)
- [use_variable_labels](#)

Validate/use variable labels from tabular file header

Specification

Alias: none

Argument(s): none

Description

When importing global surrogate training data (or challenge evaluation points) from a tabular data file containing a header (`annotated` or `custom_annotated` header), this keyword toggles validation of variable labels present in the header row.

The labels are validated against the descriptors of the variables being imported to. If the tabular file labels can be rearranged to match the expected labels, the columns in the data file will be reordered on read to match the Dakota variable order. If the read labels are not a permutation of, nor equal to, the expected labels, an error will result.

Default Behavior

When not specified, variable labels will be read, but not strictly enforced or reordered. A warning will be issued if the variable labels are not as expected, and guidance offered if they can be permuted to match expectations.

Expected Output

Console output will be generated for any warnings, as well as to indicate whether any variable reordering is taking place.

Usage Tips

The use of this keyword is recommended when importing header-annotated tabular data files where the variables are appropriately labeled. Tabular files do not always contain response labels, so no attempt is made to disambiguate variable from response labels. The variable labels must appear contiguously in the header after any leading column IDs such as 'eval_id' or 'interface'.

Examples

This example enforces variable labels for both build and challenge points

```
model
  id_model = 'SURR'
  surrogate global
    polynomial quadratic
    import_build_points_file = 'dakota_surrogate_import.unc_fixedothers.dat'
      annotated use_variable_labels
    challenge_points_file = 'dakota_surrogate_import.dat'
      annotated use_variable_labels
```

7.3.3.2.17.8 freeform

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.17.9 active_only

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_build_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.3.3.2.18 export_approx_points_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [export_approx_points_file](#)

Output file for surrogate model value evaluations

Specification

Alias: `export_points_file`

Argument(s): STRING

Default: no point export to a file

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format

Description

Specifies a file in which the points (input variable values) at which the surrogate model is evaluated and corresponding response values computed by the surrogate model will be written. The response values are the surrogate's predicted approximation to the truth model responses at those points.

Usage Tips

Dakota exports tabular data in one of three formats:

- `annotated` (default)
- `custom_annotated`
- `freeform`

7.3.3.2.18.1 custom_annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [export_approx_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn  nln_ineq_con_1  nln_ineq_con_2
1              0.9          1.1          0.0002         0.26           0.76
2              0.90009      1.1 0.0001996404857  0.2601620081    0.759955
3              0.89991      1.1 0.0002003604863  0.2598380081    0.760045
...
```

7.3.3.2.18.2 header

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.18.3 eval_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.18.4 interface_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [export_approx_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.18.5 annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [export_approx_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated_header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID            0.9         1.1         0.0002          0.26          0.76
2          NO_ID            0.90009     1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID            0.89991     1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.18.6 freeform

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [export_approx_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    freeform
```

Resulting tabular file:

```

      0.9          1.1          0.0002          0.26          0.76
0.90009        1.1 0.0001996404857  0.2601620081  0.759955
0.89991        1.1 0.0002003604863  0.2598380081  0.760045
...

```

7.3.3.2.19 use_derivatives

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [use_derivatives](#)

Use derivative data to construct surrogate models

Specification

Alias: none

Argument(s): none

Default: use function values only

Description

The `use_derivatives` flag specifies that any available derivative information should be used in global approximation builds, for those global surrogate types that support it (currently, polynomial regression and the Surpack Gaussian process).

However, it's use with Surpack Gaussian process is not recommended.

7.3.3.2.20 correction

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [correction](#)

Correction approaches for surrogate models

Specification

Alias: none

Argument(s): none

Default: no surrogate correction

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Correction Order (Group 1)	zeroth_order	Specify that truth values must be matched.
			first_order	Specify that truth values and gradients must be matched.
			second_order	Specify that truth values, gradients and Hessians must be matched.
	Required (<i>Choose One</i>)	Correction Type (Group 2)	additive	Additive correction factor for local surrogate accuracy
			multiplicative	Multiplicative correction factor for local surrogate accuracy.
			combined	Multipoint correction for a hierarchical surrogate

Description

Some of the surrogate model types support the use of correction factors that improve the local accuracy of the surrogate models.

The `correction` specification specifies that the approximation will be corrected to match truth data, either matching truth values in the case of `zeroth_order` matching, matching truth values and gradients in the case of `first_order` matching, or matching truth values, gradients, and Hessians in the case of `second_order` matching. For `additive` and `multiplicative` corrections, the correction is local in that the truth data is matched at a single point, typically the center of the approximation region. The `additive` correction adds a scalar offset (`zeroth_order`), a linear function (`first_order`), or a quadratic function (`second_order`) to the approximation to match the truth data at the point, and the `multiplicative` correction multiplies the approximation by a scalar (`zeroth_order`), a linear function (`first_order`), or a quadratic function (`second_order`) to match the truth data at the point. The `additive first_order` case is due to [59] and the `multiplicative first_order` case is commonly known as beta correction [41]. For the `combined` correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The `combined` correction is then a multipoint correction, as opposed to the local `additive` and `multiplicative` corrections. Each of these correction capabilities is described in detail in [24].

The correction factors force the surrogate models to match the true function values and possibly true function derivatives at the center point of each trust region. Currently, Dakota supports either zeroth-, first-, or second-order accurate correction methods, each of which can be applied using either an additive, multiplicative, or combined correction function. For each of these correction approaches, the correction is applied to the surrogate model and the corrected model is then interfaced with whatever algorithm is being employed. The default behavior is that no correction factor is applied.

The simplest correction approaches are those that enforce consistency in function values between the surrogate and original models at a single point in parameter space through use of a simple scalar offset or scaling applied to the surrogate model. First-order corrections such as the first-order multiplicative correction (also known as beta correction [14]) and the first-order additive correction [59] also enforce consistency in the gradients and provide a much more substantial correction capability that is sufficient for ensuring provable convergence in SBO algorithms. SBO convergence rates can be further accelerated through the use of second-order corrections which also enforce consistency in the Hessians [24], where the second-order information may involve analytic, finite-difference, or quasi-Newton Hessians.

Correcting surrogate models with additive corrections involves

$$\hat{f}_{hi\alpha}(\mathbf{x}) = f_{lo}(\mathbf{x}) + \alpha(\mathbf{x}) \quad (7.1)$$

where multifidelity notation has been adopted for clarity. For multiplicative approaches, corrections take the form

$$\hat{f}_{hi\beta}(\mathbf{x}) = f_{lo}(\mathbf{x})\beta(\mathbf{x}) \quad (7.2)$$

where, for local corrections, $\alpha(\mathbf{x})$ and $\beta(\mathbf{x})$ are first or second-order Taylor series approximations to the exact correction functions:

$$\alpha(\mathbf{x}) = A(\mathbf{x}_c) + \nabla A(\mathbf{x}_c)^T(\mathbf{x} - \mathbf{x}_c) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_c)^T \nabla^2 A(\mathbf{x}_c)(\mathbf{x} - \mathbf{x}_c) \quad (7.3)$$

$$\beta(\mathbf{x}) = B(\mathbf{x}_c) + \nabla B(\mathbf{x}_c)^T(\mathbf{x} - \mathbf{x}_c) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_c)^T \nabla^2 B(\mathbf{x}_c)(\mathbf{x} - \mathbf{x}_c) \quad (7.4)$$

where the exact correction functions are

$$A(\mathbf{x}) = f_{hi}(\mathbf{x}) - f_{lo}(\mathbf{x}) \quad (7.5)$$

$$B(\mathbf{x}) = \frac{f_{hi}(\mathbf{x})}{f_{lo}(\mathbf{x})} \quad (7.6)$$

Refer to [24] for additional details on the derivations.

A combination of additive and multiplicative corrections can provide for additional flexibility in minimizing the impact of the correction away from the trust region center. In other words, both additive and multiplicative corrections can satisfy local consistency, but through the combination, global accuracy can be addressed as well. This involves a convex combination of the additive and multiplicative corrections:

$$\hat{f}_{hi_\gamma}(\mathbf{x}) = \gamma \hat{f}_{hi_\alpha}(\mathbf{x}) + (1 - \gamma) \hat{f}_{hi_\beta}(\mathbf{x})$$

where γ is calculated to satisfy an additional matching condition, such as matching values at the previous design iterate.

It should be noted that in both first order correction methods, the function $\hat{f}(x)$ matches the function value and gradients of $f_t(x)$ at $x = x_c$. This property is necessary in proving that the first order-corrected SBO algorithms are provably convergent to a local minimum of $f_t(x)$. However, the first order correction methods are significantly more expensive than the zeroth order correction methods, since the first order methods require computing both $\nabla f_t(x_c)$ and $\nabla f_s(x_c)$. When the SBO strategy is used with either of the zeroth order correction methods, or with no correction method, convergence is not guaranteed to a local minimum of $f_t(x)$. That is, the SBO strategy becomes a heuristic optimization algorithm. From a mathematical point of view this is undesirable, but as a practical matter, the heuristic variants of SBO are often effective in finding local minima.

Usage guidelines

- Both the `additive zeroth_order` and `multiplicative zeroth_order` correction methods are "free" since they use values of $f_t(x_c)$ that are normally computed by the SBO strategy.
- The use of either the `additive first_order` method or the `multiplicative first_order` method does not necessarily improve the rate of convergence of the SBO algorithm.
- When using the first order correction methods, the gradient-related response keywords must be modified to allow either analytic or numerical gradients to be computed. This provides the gradient data needed to compute the correction function.
- For many computationally expensive engineering optimization problems, gradients often are too expensive to obtain or are discontinuous (or may not exist at all). In such cases the heuristic SBO algorithm has been an effective approach at identifying optimal designs [35].

7.3.3.2.20.1 zeroth_order

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [correction](#)
- [zeroth_order](#)

Specify that truth values must be matched.

Specification

Alias: none

Argument(s): none

Description

The correction specification specifies that the approximation will be corrected to match truth data. The keyword `zeroth_order` matching ensures that truth values are matched.

7.3.3.2.20.2 `first_order`

- [Keywords Area](#)
- `model`
- `surrogate`
- `global`
- `correction`
- `first_order`

Specify that truth values and gradients must be matched.

Specification

Alias: none

Argument(s): none

Description

This correction specification specifies that the approximation will be corrected to match truth data. The keyword `first_order` matching ensures that truth values and gradients are matched.

7.3.3.2.20.3 `second_order`

- [Keywords Area](#)
- `model`
- `surrogate`
- `global`
- `correction`
- `second_order`

Specify that truth values, gradients and Hessians must be matched.

Specification

Alias: none

Argument(s): none

Description

The correction specification specifies that the approximation will be corrected to match truth data. The keyword `second_order` matching ensures that truth values, gradients and Hessians are matched.

7.3.3.2.20.4 additive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [correction](#)
- [additive](#)

Additive correction factor for local surrogate accuracy

Specification

Alias: none

Argument(s): none

Description

Use an additive correction factor to improve the local accuracy of a surrogate.

7.3.3.2.20.5 multiplicative

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [correction](#)
- [multiplicative](#)

Multiplicative correction factor for local surrogate accuracy.

Specification

Alias: none

Argument(s): none

Description

Use a multiplicative correction factor to improve the local accuracy of a surrogate.

7.3.3.2.20.6 combined

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [global](#)
- [correction](#)
- [combined](#)

Multipoint correction for a hierarchical surrogate

Specification

Alias: none

Argument(s): none

Description

For the combined correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The combined correction is then a multipoint correction, as opposed to the local additive and multiplicative corrections.

7.3.3.2.21 metrics

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [metrics](#)

Compute surrogate quality metrics

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: diagnostics

Argument(s): STRINGLIST

Default: No diagnostics

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		cross_validation	Perform k-fold cross validation
	Optional		press	Leave-one-out cross validation

Description

Diagnostic metrics assess the goodness of fit of a global surrogate to its training data.

The default diagnostics are:

- `root_mean_squared`
- `mean_abs`
- `rsquared`

Additional available diagnostics include

- `sum_squared`
- `mean_squared`
- `sum_abs`
- `max_abs`

The keywords `press` and `cross_validation` further specify leave-one-out or k-fold cross validation, respectively, for all of the active metrics from above.

Usage Tips When specified, the `metrics` keyword must be followed by a list of quoted strings, each of which activates a metric.

Examples

This example input fragment constructs a quadratic polynomial surrogate and computes four metrics on the fit, both with and without 5-fold cross validation. (Also see `dakota/share/dakota/test/dakota_surrogate_import.in` for additional examples.)

```
model
  surrogate global
  polynomial quadratic
  metrics = "root_mean_squared" "sum_abs" "mean_abs" "max_abs"
  cross_validation folds = 5
```

Theory

Most of these diagnostics refer to some operation on the residuals (the difference between the surrogate model and the truth model at the data points upon which the surrogate is built).

For example, `sum_squared` refers to the sum of the squared residuals, and `mean_abs` refers to the mean of the absolute value of the residuals. `rsquared` refers to the R-squared value typically used in regression analysis (the proportion of the variability in the response that can be accounted for by the surrogate model). Care should be taken when interpreting metrics, for example, errors may be near zero for interpolatory models or `rsquared` may not be applicable for non-polynomial models.

7.3.3.2.2.1 `cross_validation`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [metrics](#)

- [cross_validation](#)

Perform k-fold cross validation

Topics

This keyword is related to the topics:

- [surrogate_models](#)

Specification

Alias: none

Argument(s): none

Default: No cross validation

Child Keywords:

	Required/Optional <i>Optional(Choose One)</i>	Description of Group Partition Control (Group 1)	Dakota Keyword	Dakota Keyword Description
			folds	Number of cross validation folds
			percent	Percent data per cross validation fold

Description

General k-fold cross validation may be performed by specifying `cross_validation`. The cross-validation statistics will be calculated for all metrics.

Cross validation may further specify:

- `folds`, the number of folds into which to divide the build data (between 2 and number of data points) or
- `percent`, the fraction of data (between 0 and 0.5) to use in each fold.

These will be adjusted as needed based on the number of available training points. The default number of folds $k = 10$, or 0.1

7.3.3.2.21.2 folds

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [metrics](#)
- [cross_validation](#)
- [folds](#)

Number of cross validation folds

Specification

Alias: none

Argument(s): INTEGER

Default: 10

Description

Number of folds (partitions) of the training data to use in cross validation (default 10).

7.3.3.2.21.3 percent

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [metrics](#)
- [cross_validation](#)
- [percent](#)

Percent data per cross validation fold

Specification

Alias: none

Argument(s): REAL

Default: 0.1

Description

Percent of the training data to use in each cross validation fold (default 0.1).

7.3.3.2.21.4 press

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [metrics](#)
- [press](#)

Leave-one-out cross validation

Specification

Alias: none

Argument(s): none

Default: No PRESS cross validation

Description

Leave-one-out (PRESS) cross validation may be performed by specifying `press`. The cross-validation statistics will be calculated for all metrics.

7.3.3.2.22 import_challenge_points_file

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)

Datafile of points to assess surrogate quality

Specification

Alias: challenge_points_file

Argument(s): STRING

Default: no user challenge data

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format
			annotated	Selects annotated tabular file format
			freeform	Selects freeform file format
	Optional		active_only	Import only active variables from tabular data file

Description

Specifies a data file containing variable and response (truth) values, in one of three formats:

- `annotated` (default)
- `custom_annotated`

- `freeform`

The surrogate is evaluated at the points in the file, and the surrogate (approximate) responses are compared against the truth results from the file. All metrics specified with `metrics` will be computed for the challenge data.

7.3.3.2.22.1 `custom_annotated`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		eval_id	Enable evaluation ID column in custom-annotated tabular file
	Optional		interface_id	Enable interface ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file typically containing row data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well. Custom-annotated allows user options for whether `header` row, `eval_id` column, and `interface_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

The `annotated` format is the default for tabular export/import. To control which header row and columns are in the input/output, specify `custom_annotated`, followed by options, in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a custom-annotated tabular file in Dakota 6.0 format, which contained only header and `eval_id` (no `interface_id`), and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
    custom_annotated header eval_id
```

Resulting tabular file:

```
%eval_id      x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1              0.9          1.1          0.0002          0.26          0.76
2              0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3              0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.22.2 header

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		use_variable_labels	Validate/use variable labels from tabular file header

Description

See description of parent `custom_annotated`

7.3.3.2.22.3 `use_variable_labels`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [custom_annotated](#)
- [header](#)
- [use_variable_labels](#)

Validate/use variable labels from tabular file header

Specification

Alias: none

Argument(s): none

Description

When importing global surrogate training data (or challenge evaluation points) from a tabular data file containing a header (`annotated` or `custom_annotated` header), this keyword toggles validation of variable labels present in the header row.

The labels are validated against the descriptors of the variables being imported to. If the tabular file labels can be rearranged to match the expected labels, the columns in the data file will be reordered on read to match the Dakota variable order. If the read labels are not a permutation of, nor equal to, the expected labels, an error will result.

Default Behavior

When not specified, variable labels will be read, but not strictly enforced or reordered. A warning will be issued if the variable labels are not as expected, and guidance offered if they can be permuted to match expectations.

Expected Output

Console output will be generated for any warnings, as well as to indicate whether any variable reordering is taking place.

Usage Tips

The use of this keyword is recommended when importing header-annotated tabular data files where the variables are appropriately labeled. Tabular files do not always contain response labels, so no attempt is made to disambiguate variable from response labels. The variable labels must appear contiguously in the header after any leading column IDs such as `'eval_id'` or `'interface'`.

Examples

This example enforces variable labels for both build and challenge points

```
model
  id_model = 'SURR'
  surrogate global
    polynomial quadratic
  import_build_points_file = 'dakota_surrogate_import_unc_fixedothers.dat'
    annotated use_variable_labels
  challenge_points_file = 'dakota_surrogate_import.dat'
    annotated use_variable_labels
```

7.3.3.2.22.4 eval_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [custom_annotated](#)
- [eval_id](#)

Enable evaluation ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.22.5 interface_id

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [custom_annotated](#)
- [interface_id](#)

Enable interface ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Description

See description of parent `custom_annotated`

7.3.3.2.22.6 annotated

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [annotated](#)

Selects annotated tabular file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		use_variable_labels	Validate/use variable labels from tabular file header

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. Each subsequent row contains an evaluation ID and interface ID, followed by data for variables, or variables followed by responses, depending on context.

Default Behavior

By default, Dakota imports and exports tabular files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- To specify pre-Dakota 6.1 tabular format, which did not include `interface_id`, specify `custom_annotated header eval_id`
- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.

- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export an annotated top-level tabular data file containing a header row, leading `eval_id` and `interface_id` columns, and data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  annotated
```

Resulting tabular file:

```
%eval_id interface          x1          x2          obj_fn nln_ineq_con_1 nln_ineq_con_2
1          NO_ID           0.9          1.1          0.0002          0.26          0.76
2          NO_ID           0.90009      1.1 0.0001996404857 0.2601620081 0.759955
3          NO_ID           0.89991      1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.22.7 use_variable_labels

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [annotated](#)
- [use_variable_labels](#)

Validate/use variable labels from tabular file header

Specification

Alias: none

Argument(s): none

Description

When importing global surrogate training data (or challenge evaluation points) from a tabular data file containing a header (`annotated` or `custom_annotated` header), this keyword toggles validation of variable labels present in the header row.

The labels are validated against the descriptors of the variables being imported to. If the tabular file labels can be rearranged to match the expected labels, the columns in the data file will be reordered on read to match the Dakota variable order. If the read labels are not a permutation of, nor equal to, the expected labels, an error will result.

Default Behavior

When not specified, variable labels will be read, but not strictly enforced or reordered. A warning will be issued if the variable labels are not as expected, and guidance offered if they can be permuted to match expectations.

Expected Output

Console output will be generated for any warnings, as well as to indicate whether any variable reordering is taking place.

Usage Tips

The use of this keyword is recommended when importing header-annotated tabular data files where the variables are appropriately labeled. Tabular files do not always contain response labels, so no attempt is made to disambiguate variable from response labels. The variable labels must appear contiguously in the header after any leading column IDs such as 'eval_id' or 'interface'.

Examples

This example enforces variable labels for both build and challenge points

```
model
  id_model = 'SURR'
  surrogate global
    polynomial quadratic
    import_build_points_file = 'dakota_surrogate_import.unc_fixedothers.dat'
      annotated use_variable_labels
    challenge_points_file = 'dakota_surrogate_import.dat'
      annotated use_variable_labels
```

7.3.3.2.22.8 freeform

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [freeform](#)

Selects freeform file format

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. Most commonly, each row contains data for variables, or variables followed by responses, though the format is used for other tabular exports/imports as well.

Default Behavior

The `annotated` format is the default for tabular export/import. To change this behavior, specify `freeform` in the relevant export/import context.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were free-form format. They now default to `annotated` format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.
- In `freeform`, the `num_rows` x `num_cols` total data entries may be separated with any whitespace including spaces, tabs, and newlines. In this format, vectors may therefore appear as a single row or single column (or mixture; entries will populate the vector in order).
- Some TPLs like SCOLIB and JEGA manage their own file I/O and only support the `freeform` option.

Examples

Export a freeform tabular file containing only data for variables and responses. Input file fragment:

```
environment
  tabular_data
    tabular_data_file = 'dakota_summary.dat'
  freeform
```

Resulting tabular file:

```
      0.9          1.1          0.0002          0.26          0.76
0.90009          1.1 0.0001996404857 0.2601620081 0.759955
0.89991          1.1 0.0002003604863 0.2598380081 0.760045
...
```

7.3.3.2.22.9 active_only

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [global](#)
- [import_challenge_points_file](#)
- [active_only](#)

Import only active variables from tabular data file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Description

By default, files for tabular data imports are expected to contain columns for all variables, active and inactive. The keyword `active_only` indicates that the file to import contains only the active variables.

This option should only be used in contexts where the inactive variables have no influence, for example, building a surrogate over active variables, with the state variables held at nominal. It should not be used in more complex nested contexts, where the values of inactive variables are relevant to the function evaluations used to build the surrogate.

7.3.3.3 multipoint

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [multipoint](#)

Construct a surrogate from multiple existing training points

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional <i>Required (Choose One)</i>	Description of Group Multipoint Surrogate (Group 1)	Dakota Keyword tana	Dakota Keyword Description Local multi-point model via two-point nonlinear approximation
			qmea	Multi-point surrogate approximation based on QMEA algorithm
	Required		truth_model_pointer	Pointer to specify a "truth" model, from which to construct a surrogate

Description

Multipoint approximations use data from previous design points to improve the accuracy of local approximations. The data often comes from the current and previous iterates of a minimization algorithm.

Currently, only the Two-point Adaptive Nonlinearity Approximation (TANA-3) method of [95] is supported with the `tana` keyword.

The truth model to be used to generate the value/gradients used in the approximation is identified through the required `truth_model_pointer` specification.

See Also

These keywords may also be of interest:

- [local](#)
- [global](#)
- [hierarchical](#)

7.3.3.3.1 tana

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [multipoint](#)
- [tana](#)

Local multi-point model via two-point nonlinear approximation

Specification

Alias: none

Argument(s): none

Description

TANA stands for Two Point Adaptive Nonlinearity Approximation.

The TANA-3 method [95] is a multipoint approximation method based on the two point exponential approximation [25]. This approach involves a Taylor series approximation in intermediate variables where the powers used for the intermediate variables are selected to match information at the current and previous expansion points.

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Theory

The form of the TANA model is:

$$\hat{f}(\mathbf{x}) \approx f(\mathbf{x}_2) + \sum_{i=1}^n \frac{\partial f}{\partial x_i}(\mathbf{x}_2) \frac{x_{i,2}^{1-p_i}}{p_i} (x_i^{p_i} - x_{i,2}^{p_i}) + \frac{1}{2} \varepsilon(\mathbf{x}) \sum_{i=1}^n (x_i^{p_i} - x_{i,2}^{p_i})^2$$

where n is the number of variables and:

$$p_i = 1 + \ln \left[\frac{\frac{\partial f}{\partial x_i}(\mathbf{x}_1)}{\frac{\partial f}{\partial x_i}(\mathbf{x}_2)} \right] \bigg/ \ln \left[\frac{x_{i,1}}{x_{i,2}} \right] \quad \varepsilon(\mathbf{x}) = \frac{H}{\sum_{i=1}^n (x_i^{p_i} - x_{i,1}^{p_i})^2 + \sum_{i=1}^n (x_i^{p_i} - x_{i,2}^{p_i})^2} H = 2 \left[f(\mathbf{x}_1) - f(\mathbf{x}_2) - \sum_{i=1}^n \frac{\partial f}{\partial x_i}(\mathbf{x}_2) \frac{x_{i,2}^{1-p_i}}{p_i} (x_{i,1}^{p_i} - x_{i,2}^{p_i}) \right]$$

and \mathbf{x}_2 and \mathbf{x}_1 are the current and previous expansion points. Prior to the availability of two expansion points, a first-order Taylor series is used.

7.3.3.3.2 qmea

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [multipoint](#)
- [qmea](#)

Multi-point surrogate approximation based on QMEA algorithm

Specification

Alias: none

Argument(s): none

Description

The Quadratic Multipoint Exponential Approximation (QMEA) builds a multi-point approximation from values and gradients at multiple expansion points. It is a generalization of the two-point exponential approximation (TPEA).

This capability is **experimental**.

7.3.3.3.3 truth_model_pointer

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [multipoint](#)
- [truth_model_pointer](#)

Pointer to specify a "truth" model, from which to construct a surrogate

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: actual_model_pointer

Argument(s): STRING

Description

This must point to a model block, identified by [id_model](#). That model will be run to generate training data, from which a surrogate model will be constructed.

See [block_pointer](#) for details about pointers.

7.3.3.4 local

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [local](#)

Build a locally accurate surrogate from data at a single point

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		taylor_series	Construct a Taylor Series expansion around a point
	Required		truth_model_pointer	Pointer to specify a "truth" model, from which to construct a surrogate

Description

Local approximations use value, gradient, and possibly Hessian data from a single point to form a series expansion for approximating data in the vicinity of this point.

The currently available local approximation is the `taylor_series` selection.

The truth model to be used to generate the value/gradient/Hessian data used in the series expansion is identified through the required `truth_model_pointer` specification. The use of a model pointer (as opposed to an interface pointer) allows additional flexibility in defining the approximation. In particular, the derivative specification for the truth model may differ from the derivative specification for the approximation, and the truth model results being approximated may involve a model recursion (e.g., the values/gradients from a nested model).

See Also

These keywords may also be of interest:

- [global](#)
- [hierarchical](#)
- [multipoint](#)

7.3.3.4.1 `taylor_series`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)

- [local](#)
- [taylor_series](#)

Construct a Taylor Series expansion around a point

Specification

Alias: none

Argument(s): none

Description

The Taylor series model is purely a local approximation method. That is, it provides local trends in the vicinity of a single point in parameter space.

The order of the Taylor series may be either first-order or second-order, which is automatically determined from the gradient and Hessian specifications in the responses specification (see [responses](#) for info on how to specify gradients and Hessians) for the truth model

Known Issue: When using discrete variables, there have been sometimes significant differences in surrogate behavior observed across computing platforms in some cases. The cause has not yet been fully diagnosed and is currently under investigation. In addition, guidance on appropriate construction and use of surrogates with discrete variables is under development. In the meantime, users should therefore be aware that there is a risk of inaccurate results when using surrogates with discrete variables.

Theory

The first-order Taylor series expansion is:

$$\hat{f}(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla_{\mathbf{x}}f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) \quad (7.7)$$

and the second-order expansion is:

$$\hat{f}(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla_{\mathbf{x}}f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla_{\mathbf{x}}^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \quad (7.8)$$

where \mathbf{x}_0 is the expansion point in n -dimensional parameter space and $f(\mathbf{x}_0)$, $\nabla_{\mathbf{x}}f(\mathbf{x}_0)$, and $\nabla_{\mathbf{x}}^2 f(\mathbf{x}_0)$ are the computed response value, gradient, and Hessian at the expansion point, respectively.

As dictated by the responses specification used in building the local surrogate, the gradient may be analytic or numerical and the Hessian may be analytic, numerical, or based on quasi-Newton secant updates.

In general, the Taylor series model is accurate only in the region of parameter space that is close to \mathbf{x}_0 . While the accuracy is limited, the first-order Taylor series model reproduces the correct value and gradient at the point \mathbf{x}_0 , and the second-order Taylor series model reproduces the correct value, gradient, and Hessian. This consistency is useful in provably-convergent surrogate-based optimization. The other surface fitting methods do not use gradient information directly in their models, and these methods rely on an external correction procedure in order to satisfy the consistency requirements of provably-convergent SBO.

7.3.3.4.2 truth_model_pointer

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [local](#)

- [truth_model_pointer](#)

Pointer to specify a "truth" model, from which to construct a surrogate

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: actual_model_pointer

Argument(s): STRING

Description

This must point to a model block, identified by [id_model](#). That model will be run to generate training data, from which a surrogate model will be constructed.

See [block_pointer](#) for details about pointers.

7.3.3.5 hierarchical

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)

Hierarchical surrogates use an ordered hierarchy of model fidelities and/or resolutions to trade accuracy versus cost.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		ordered_model_fidelities	Specification of an hierarchy of model fidelities, ordered from low to high.
	Optional		correction	Correction approaches for surrogate models

Description

Multifidelity modeling

involves the use of lower-fidelity physics-based models as surrogates for a high-fidelity "truth" model. These low-fidelity models can involve variations in model form, resolution, or both. Model form variations for reduced fidelity may involve omitted physics or solution of approximated or averaged physics. Resolution variations typically involve coarsened space-time discretizations (e.g., h- derefinement), reduced element order (e.g., p- derefinement), or loosened convergence tolerances. Unlike local, global, and multipoint surrogates, these approximations are not data-driven (no high-fidelity data required for construction) and are stand-alone simulation models in their own right. As an example from computational fluid dynamics (CFD), both model form and resolution are varied when an inviscid, incompressible Euler model on a coarse discretization is used as a low-fidelity surrogate for a high-fidelity large eddy simulation model on a fine discretization.

The required `ordered_model_fidelities` specification points to a sequence of model specifications of varying fidelity, ordered from lowest to highest fidelity. The highest fidelity model provides the "truth" model, and each of the lower fidelity alternatives provides different levels of approximation at different levels of cost. This specification defines the sequence of model forms, and each model specification identified in this ordered listing may identify a set of resolution controls. Either or both of these sequences may be specified, and this is referred to as a "multilevel hierarchy" in the case of a hierarchy of resolutions (one entry in `ordered_model_fidelities` that includes active resolution control), a "multifidelity hierarchy" in the case of a hierarchy of model forms (multiple entries in `ordered_model_fidelities` without active resolution control), or a "multilevel-multifidelity hierarchy" in the case of two-dimensional hierarchy including both model forms and resolutions (multiple entries in `ordered_model_fidelities` with one or more including active resolution control). Note that the multilevel-multifidelity case can be "ragged" in the sense that not all models need to provide the same number of (or any) resolution controls.

The `correction` specification identifies what type (`additive`, `multiplicative`, `combined`) and order (`zeroth_order`, `first_order`, `second_order`) of correction technique will be applied to the low fidelity results in order to match high fidelity results (value and potentially gradient and Hessian) at one or more points. As described below, this is essential in the optimization context.

Use cases

In multifidelity surrogate-based optimization (SBO), the search algorithm relies primarily on the lower fidelity models, which are corrected for consistency with higher fidelity models. The higher fidelity models are used primarily for verifying candidate steps based on solution of low fidelity approximate subproblems and for updating low fidelity corrections. In the hierarchical SBO case (as compared to SBO with data fits), the `correction` specification is required, since the omission of a correction technique would effectively eliminate the purpose of the high fidelity model (to use a low fidelity model without corrections, then a `single` model can be used rather than a `hierarchical` model). Refer to [global](#) for additional information on available correction approaches.

In multifidelity uncertainty quantification (UQ), response differences are tracked for purposes of decomposing variance across model/resolution levels or for constructing separate discrepancy emulators. In this context, correction specifications are still valid for defining discrepancy emulation details but they are optional with the most common cases used as defaults.

Examples

Theory:

See Also

These keywords may also be of interest:

- [global](#)
- [local](#)
- [multipoint](#)

- [multilevel_sampling](#)
- [multilevel_polynomial_chaos](#)
- [multifidelity_polynomial_chaos](#)
- [multifidelity_stoch_collocation](#)
- [multilevel_function_train](#)
- [multifidelity_function_train](#)
- [surrogate_based_local](#)

7.3.3.5.1 ordered_model_fidelities

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)
- [ordered_model_fidelities](#)

Specification of an hierarchy of model fidelities, ordered from low to high.

Specification

Alias: model_fidelity_sequence

Argument(s): STRINGLIST

Description

A hierarchical surrogate model manages an ordered set of model fidelities, each of which may in turn involve multiple discretization levels (in the case of a simulation model) or additional model recursions.

The ordering is assumed to be from lowest fidelity to highest fidelity, as dictated by an accuracy versus cost trade-off. Corresponding sequence specifications within methods (e.g., `quadrature_order_sequence`, `sparse_grid_level_sequence`, `expansion_order_sequence`, etc. within stochastic expansion methods) should be synchronized with this model order.

Additional Discussion

Internal to the model, only one low fidelity model instance and one high fidelity model instance are active at any given time, although various optimization and UQ algorithms can be used to traverse deep multilevel and multifidelity hierarchies by activating different model combinations and different response modes within the hierarchical model infrastructure.

Examples

```
model,
  id_model = 'HIERARCH'
  surrogate hierarchical
    ordered_model_fidelities = 'LF' 'MF' 'HF'
    correction additive zeroth_order

model,
  id_model = 'LF'
  simulation
    interface_pointer = 'LF_DRIVER'
```

```

model,
  id_model = 'MF'
  simulation
    interface_pointer = 'MF_DRIVER'

model,
  id_model = 'HF'
  simulation
    interface_pointer = 'HF_DRIVER'

```

See Also

These keywords may also be of interest:

- [multilevel_sampling](#)
- [polynomial_chaos](#)
- [stoch_collocation](#)
- [surrogate_based_local](#)

7.3.3.5.2 correction

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)
- [correction](#)

Correction approaches for surrogate models

Specification

Alias: none

Argument(s): none

Default: no surrogate correction

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Correction Order (Group 1)	zeroth_order	Specify that truth values must be matched.
			first_order	Specify that truth values and gradients must be matched.

			<code>second_order</code>	Specify that truth values, gradients and Hessians must be matched.
	Required (Choose One)	Correction Type (Group 2)	<code>additive</code>	Additive correction factor for local surrogate accuracy
			<code>multiplicative</code>	Multiplicative correction factor for local surrogate accuracy.
			<code>combined</code>	Multipoint correction for a hierarchical surrogate

Description

Some of the surrogate model types support the use of correction factors that improve the local accuracy of the surrogate models.

The `correction` specification specifies that the approximation will be corrected to match truth data, either matching truth values in the case of `zeroth_order` matching, matching truth values and gradients in the case of `first_order` matching, or matching truth values, gradients, and Hessians in the case of `second_order` matching. For `additive` and `multiplicative` corrections, the correction is local in that the truth data is matched at a single point, typically the center of the approximation region. The `additive` correction adds a scalar offset (`zeroth_order`), a linear function (`first_order`), or a quadratic function (`second_order`) to the approximation to match the truth data at the point, and the `multiplicative` correction multiplies the approximation by a scalar (`zeroth_order`), a linear function (`first_order`), or a quadratic function (`second_order`) to match the truth data at the point. The `additive first_order` case is due to [59] and the `multiplicative first_order` case is commonly known as beta correction [41]. For the `combined` correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The `combined` correction is then a multipoint correction, as opposed to the local `additive` and `multiplicative` corrections. Each of these correction capabilities is described in detail in [24].

The correction factors force the surrogate models to match the true function values and possibly true function derivatives at the center point of each trust region. Currently, Dakota supports either zeroth-, first-, or second-order accurate correction methods, each of which can be applied using either an additive, multiplicative, or combined correction function. For each of these correction approaches, the correction is applied to the surrogate model and the corrected model is then interfaced with whatever algorithm is being employed. The default behavior is that no correction factor is applied.

The simplest correction approaches are those that enforce consistency in function values between the surrogate and original models at a single point in parameter space through use of a simple scalar offset or scaling applied to the surrogate model. First-order corrections such as the first-order multiplicative correction (also known as beta correction [14]) and the first-order additive correction [59] also enforce consistency in the gradients and provide a much more substantial correction capability that is sufficient for ensuring provable convergence in SBO algorithms. SBO convergence rates can be further accelerated through the use of second-order corrections which also enforce consistency in the Hessians [24], where the second-order information may involve analytic, finite-difference, or quasi-Newton Hessians.

Correcting surrogate models with additive corrections involves

$$\hat{f}_{hi\alpha}(\mathbf{x}) = f_{io}(\mathbf{x}) + \alpha(\mathbf{x}) \quad (7.9)$$

where multifidelity notation has been adopted for clarity. For multiplicative approaches, corrections take the form

$$\hat{f}_{hi\beta}(\mathbf{x}) = f_{lo}(\mathbf{x})\beta(\mathbf{x}) \quad (7.10)$$

where, for local corrections, $\alpha(\mathbf{x})$ and $\beta(\mathbf{x})$ are first or second-order Taylor series approximations to the exact correction functions:

$$\alpha(\mathbf{x}) = A(\mathbf{x}_c) + \nabla A(\mathbf{x}_c)^T(\mathbf{x} - \mathbf{x}_c) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_c)^T \nabla^2 A(\mathbf{x}_c)(\mathbf{x} - \mathbf{x}_c) \quad (7.11)$$

$$\beta(\mathbf{x}) = B(\mathbf{x}_c) + \nabla B(\mathbf{x}_c)^T(\mathbf{x} - \mathbf{x}_c) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_c)^T \nabla^2 B(\mathbf{x}_c)(\mathbf{x} - \mathbf{x}_c) \quad (7.12)$$

where the exact correction functions are

$$A(\mathbf{x}) = f_{hi}(\mathbf{x}) - f_{lo}(\mathbf{x}) \quad (7.13)$$

$$B(\mathbf{x}) = \frac{f_{hi}(\mathbf{x})}{f_{lo}(\mathbf{x})} \quad (7.14)$$

Refer to [24] for additional details on the derivations.

A combination of additive and multiplicative corrections can provide for additional flexibility in minimizing the impact of the correction away from the trust region center. In other words, both additive and multiplicative corrections can satisfy local consistency, but through the combination, global accuracy can be addressed as well. This involves a convex combination of the additive and multiplicative corrections:

$$\hat{f}_{hi\gamma}(\mathbf{x}) = \gamma \hat{f}_{hi\alpha}(\mathbf{x}) + (1 - \gamma) \hat{f}_{hi\beta}(\mathbf{x})$$

where γ is calculated to satisfy an additional matching condition, such as matching values at the previous design iterate.

It should be noted that in both first order correction methods, the function $\hat{f}(x)$ matches the function value and gradients of $f_t(x)$ at $x = x_c$. This property is necessary in proving that the first order-corrected SBO algorithms are provably convergent to a local minimum of $f_t(x)$. However, the first order correction methods are significantly more expensive than the zeroth order correction methods, since the first order methods require computing both $\nabla f_t(x_c)$ and $\nabla f_s(x_c)$. When the SBO strategy is used with either of the zeroth order correction methods, or with no correction method, convergence is not guaranteed to a local minimum of $f_t(x)$. That is, the SBO strategy becomes a heuristic optimization algorithm. From a mathematical point of view this is undesirable, but as a practical matter, the heuristic variants of SBO are often effective in finding local minima.

Usage guidelines

- Both the `additive zeroth_order` and `multiplicative zeroth_order` correction methods are "free" since they use values of $f_t(x_c)$ that are normally computed by the SBO strategy.
- The use of either the `additive first_order` method or the `multiplicative first_order` method does not necessarily improve the rate of convergence of the SBO algorithm.
- When using the first order correction methods, the gradient-related response keywords must be modified to allow either analytic or numerical gradients to be computed. This provides the gradient data needed to compute the correction function.
- For many computationally expensive engineering optimization problems, gradients often are too expensive to obtain or are discontinuous (or may not exist at all). In such cases the heuristic SBO algorithm has been an effective approach at identifying optimal designs [35].

7.3.3.5.2.1 zeroth_order

- [Keywords Area](#)

- [model](#)
- [surrogate](#)
- [hierarchical](#)
- [correction](#)
- [zeroth_order](#)

Specify that truth values must be matched.

Specification

Alias: none

Argument(s): none

Description

The correction specification specifies that the approximation will be corrected to match truth data. The keyword `zeroth_order` matching ensures that truth values are matched.

7.3.3.5.2.2 `first_order`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)
- [correction](#)
- [first_order](#)

Specify that truth values and gradients must be matched.

Specification

Alias: none

Argument(s): none

Description

This correction specification specifies that the approximation will be corrected to match truth data. The keyword `first_order` matching ensures that truth values and gradients are matched.

7.3.3.5.2.3 `second_order`

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)

- [correction](#)
- [second_order](#)

Specify that truth values, gradients and Hessians must be matched.

Specification

Alias: none

Argument(s): none

Description

The correction specification specifies that the approximation will be corrected to match truth data. The keyword `second_order` matching ensures that truth values, gradients and Hessians are matched.

7.3.3.5.2.4 additive

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)
- [correction](#)
- [additive](#)

Additive correction factor for local surrogate accuracy

Specification

Alias: none

Argument(s): none

Description

Use an additive correction factor to improve the local accuracy of a surrogate.

7.3.3.5.2.5 multiplicative

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)
- [correction](#)
- [multiplicative](#)

Multiplicative correction factor for local surrogate accuracy.

Specification

Alias: none

Argument(s): none

Description

Use a multiplicative correction factor to improve the local accuracy of a surrogate.

7.3.3.5.2.6 combined

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [hierarchical](#)
- [correction](#)
- [combined](#)

Multipoint correction for a hierarchical surrogate

Specification

Alias: none

Argument(s): none

Description

For the combined correction, the use of both additive and multiplicative corrections allows the satisfaction of an additional matching condition, typically the truth function values at the previous correction point (e.g., the center of the previous trust region). The combined correction is then a multipoint correction, as opposed to the local additive and multiplicative corrections.

7.3.3.6 non_hierarchical

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [non_hierarchical](#)

Non-hierarchical surrogates employ an unordered set of lower-fidelity models to approximate a truth reference model at reduced cost.

Specification

Alias: model_ensemble

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		truth_model_pointer	Pointer to specify a "truth" model, from which to construct a surrogate
	Optional		unordered_model_fidelities	Specification of an unordered ensemble of low-fidelity approximations

See Also

These keywords may also be of interest:

- [hierarchical](#)

7.3.3.6.1 truth_model_pointer

- [Keywords Area](#)
- [model](#)
- [surrogate](#)
- [non_hierarchical](#)
- [truth_model_pointer](#)

Pointer to specify a "truth" model, from which to construct a surrogate

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: actual_model_pointer

Argument(s): STRING

Description

This must point to a model block, identified by [id_model](#). That model will be run to generate training data, from which a surrogate model will be constructed.

See [block_pointer](#) for details about pointers.

7.3.3.6.2 unordered_model_fidelities

- [Keywords Area](#)
- [model](#)

- [surrogate](#)
- [non_hierarchical](#)
- [unordered_model_fidelities](#)

Specification of an unordered ensemble of low-fidelity approximations

Specification

Alias: `approximation_models`

Argument(s): STRINGLIST

Description

A `non_hierarchical` surrogate model manages an unordered set of low-fidelity model approximations, each of which may include hyper-parameter resolution controls (in the case of a simulation model) or additional model recursions.

Any corresponding sequence specifications within methods (e.g., `quadrature_order_sequence`, `sparse_grid_level_sequence`, `expansion_order_sequence`, etc. within stochastic expansion methods) should be synchronized with the order in the model listing.

Internal to the non-hierarchical model, subsets of the model ensemble may be active for any given evaluation, as dictated by the iterative algorithm in use.

Examples

```
model,  
  id_model = 'NONHIERARCH'  
  surrogate non_hierarchical  
    unordered_model_fidelities = 'LF1' 'LF2'  
  truth_model_pointer = 'HF'
```

```
model,  
  id_model = 'LF1'  
  simulation  
    interface_pointer = 'LF1_DRIVER'  
    solution_level_cost = 1.
```

```
model,  
  id_model = 'LF2'  
  simulation  
    interface_pointer = 'LF2_DRIVER'  
    solution_level_cost = 2.4
```

```
model,  
  id_model = 'HF'  
  simulation  
    interface_pointer = 'HF_DRIVER'  
    solution_level_cost = 256.
```

See Also

These keywords may also be of interest:

- [hierarchical](#)

7.3.4 nested

- [Keywords Area](#)
- [model](#)
- [nested](#)

A model whose responses are computed through the use of a sub-iterator

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		optional_interface_pointer	Pointer to interface that provides non-nested responses
	Required		sub_method_pointer	The <code>sub_method_pointer</code> specifies the method block for the sub-iterator

Description

Instead of appealing directly to a primary interface, a nested model maps variables to responses by executing a secondary iterator, or a "sub-iterator". In other words, a function evaluation of the primary study consists of a solution of an entire secondary study - potentially many secondary function evaluations.

The sub-iterator in turn operates on a sub-model. The sub-iterator responses may be combined with non-nested contributions from an optional interface specification.

A **`sub_method_pointer`** must be provided in order to specify the method block describing the sub-iterator. The remainder of the model is specified under that keyword.

A **`optional_interface_pointer`** points to the interface specification and `optional_interface_responses_pointer` points to a responses specification describing the data to be returned by this interface. This interface is used to provide non-nested data, which is then combined with data from the nested iterator using the `primary_response_mapping`. (See the discussions for the `sub_method_pointer` and `optional_interface_pointer` keywords.)

See Also

These keywords may also be of interest:

- [single](#)
- [surrogate](#)

7.3.4.1 optional_interface_pointer

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [optional_interface_pointer](#)

Pointer to interface that provides non-nested responses

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: no optional interface

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		optional_interface-responses_pointer	Pointer to responses block that defines non-nested responses

Description

`optional_interface_pointer` is used to specify an optional interface (using that interface block's `id_interface` label) to provide non-nested responses to the nested model.

To generate the complete set of primary responses (response functions, objective functions, and calibration terms) expected by the nested model, primary responses from the optional interface may be combined (by addition) with responses derived from sub-iterator results, or they may be presented unchanged to the nested model. The way that Dakota treats primary responses from these two sources depends on the number of rows of and entries in the `primary_response_mapping` matrix.

To understand how, it is helpful to consider the following expression. Nested model primary responses r are computed from the column vector of optional interface responses f , the `primary_response_mapping` matrix W , and the column vector of results from the sub-iterator s .

$$\{r\} = \left\{ \begin{array}{c} f \\ 0 \end{array} \right\} + \left\{ \begin{array}{c} [W]\{s\} \\ 0 \end{array} \right\}$$

The number of rows in f is equal to the number of primary responses returned by the optional interface, and in r , the number of primary responses expected by the nested model. In this expression, 0 represents "padding" that may be needed by either the optional interface vector or the sub-iterator vector to make them the same dimension as r . One or the other of these quantities may be padded, but not both. In terms of the Dakota input, this means that the number of nested model primary responses must equal the larger of these two quantities: the number of optional interface responses or the number of rows in the `primary_response_mapping`.

The nested model's secondary responses (nonlinear constraints) are the secondary responses from the optional interface augmented by the secondary responses constructed from sub-iterator results. Unlike primary responses from these two sources, which may be combined by adding, it is not possible presently to combine secondary responses. For this reason, the number of nested model's secondary responses must equal the number of secondary responses provided by the optional interface, plus the number provided by the sub-iterator (i.e. the number of rows in the `secondary_response_mapping`). Secondary responses from the optional interface are inserted first into the nested model's nonlinear constraints; those from the sub-iterator come after. If the nested model has secondary responses and an optional interface is used, it is necessary to specify a separate response block for the optional interface using the `optional_interface_response_pointer`. Targets and upper and lower bounds for the nonlinear constraints are taken from the nested model's response block.

Examples

The first example illustrates the use of an optional interface and primary response mapping. No secondary responses are present. Assume that the inner method returns two results (suppose its the mean and standard deviation of a single response), so that the dimensions of `primary_response_mapping` will be interpreted by Dakota as 1x2, just as they are written here.

The nested model (`nested`) expects two primary responses (`outer_resp` block), and the optional interface returns two (`opt_resp` block).

```
model
  id_model 'nested'
  nested
    optional_interface_pointer 'opt_intf'
    optional_interface_response_pointer 'opt_resp'
    sub_method_pointer 'inner_method'
    primary_response_mapping 1.0 2.0

    responses_pointer 'outer_resp'
    variables_pointer 'outer_vars'

responses
  id_responses 'opt_resp'
  response_functions 2
  descriptors 'opt1' 'opt2'
  no_gradients
  no_hessians

responses
  id_responses 'outer_resp'
  response_functions 2
  descriptors 'out1' 'out2'
  no_gradients
  no_hessians
```

The first nested model response, `out1` will be computed $out1 = opt1 + 1.0 * mean + 2.0 * stddev$, and the second response `out2` will be equal to simply `opt2`. In the matrix form introduced above, including the implicit padding:

$$\begin{Bmatrix} out1 \\ out2 \end{Bmatrix} = \begin{Bmatrix} opt1 \\ opt2 \end{Bmatrix} + \begin{Bmatrix} 1.0 \cdot mean + 2.0 \cdot stddev \\ 0 \end{Bmatrix}$$

The `optional_interface_response_pointer` was not strictly needed. Because the number of responses returned by the optional interface and expected by the nested model is the same (and no secondary responses are present), the nested model's response specification alone would have sufficed.

If it were desired to combine `opt2` with the sub-iterator's results and set `out1` to `opt1` with no combination, then the `primary_response_mapping` matrix could be changed to:

```
primary_response_mapping 0.0 0.0
                        1.0 2.0
```

Secondary responses will be demonstrated in the second example.


```

model
  id_model 'nested'
  nested
    optional_interface_pointer 'opt_intf'
    optional_interface_response_pointer 'opt_resp'
    sub_method_pointer 'inner_method'
    primary_response_mapping 1.0 2.0
    secondary_response_mapping 1.0 0.0
    responses_pointer 'outer_resp'
    variables_pointer 'outer_vars'

responses
  id_responses 'opt_resp'
  objective_functions 2
    descriptors 'opt1' 'opt2'
  nonlinear_inequality_constraints 1
    descriptors 'optc'
  no_gradients
  no_hessians

responses
  id_responses 'outer_resp'
  objective_functions 2
    descriptors 'out1' 'out2'
  nonlinear_inequality_constraints 2
    descriptors 'outc1' 'outc2'
  no_gradients
  no_hessians

```

The nested model expects a total of two constraints. Because the `secondary_response_mapping` has one row, one of these constraints will be supplied from the sub-iterator. The other is provided by the optional interface. Secondary responses from the optional iterator come first, and so `optc` is placed in `outc1`. `outc2` takes the value computed from the sub-iterator results.

7.3.4.1.1 optional_interface_responses_pointer

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [optional_interface_pointer](#)
- [optional_interface_responses_pointer](#)

Pointer to responses block that defines non-nested responses

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: reuse of top-level responses specification

Description

`optional_interface_responses_pointer` points to the responses block (specifically, its [id_responses](#) label) that defines the non-nested response to return to the nested model. The `primary_response_mapping` keywords control how these non-nested responses are combined with responses from the nested sub-iterator. See the entry for `optional_interface_pointer` for a full description. If `optional_interface_responses_pointer` is not provided, the top-level `responses` specification is reused.

7.3.4.2 sub_method_pointer

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)

The `sub_method_pointer` specifies the method block for the sub-iterator

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		iterator_servers	Specify the number of iterator servers when Dakota is run in parallel
	Optional		iterator_scheduling	Specify the scheduling of concurrent iterators when Dakota is run in parallel
	Optional		processors_per_iterator	Specify the number of processors per iterator server when Dakota is run in parallel

	Optional	primary_variable_-mapping	Primary mapping of top-level variables to sub-model variables
	Optional	secondary_variable_mapping	Secondary mapping of top-level variables to sub-model variables
	Optional	primary_response_-mapping	Mapping of sub-method results to top-level primary responses
	Optional	secondary_response_mapping	Mapping of sub-method results to top-level secondary responses
	Optional	identity_response_-mapping	Identity mapping of sub-method results to top-level responses

Description

The `sub_method_pointer` specifies the method block for the sub-iterator.

See [block_pointer](#) for details about pointers.

A nested model performs an evaluation (variables-to-responses mapping) by running a complete sub-iterator (i.e. method) to completion. The sub-iterator, which is specified using the `sub_method_pointer`, has its own model. This model, referred to here as the sub-model, possesses its own variables and responses. Prior to every execution of the sub-iterator, information about the nested model's variables is injected into the sub-model's variables. After the sub-iterator has completed, its results are passed back up to the nested model as responses.

Communication of variables information from a nested model to its sub-iterator, and in the opposite direction of sub-iterator results to nested model responses, is called mapping. (This mapping must not be confused with the variables-to-responses mapping that constitutes an evaluation.) Dakota allows considerable power and flexibility in how nested model mappings are performed. They are specified using four keywords: for the variables, `primary_variable_mapping` and `secondary_variable_mapping`; and for the responses, `primary_response_mapping` and `secondary_response_mapping`. They are described below.

Variable Mappings

In the variable mapping case, primary and secondary variable mapping specifications are used to map from the top-level variables into the sub-model variables. These mappings support three possibilities in any combination: (1) insertion of an active top-level variable value into an identified sub-model distribution parameter for an identified active sub-model variable, (2) insertion of an active top-level variable value into an identified active sub-model variable value, and (3) addition of an active top-level variable value as an inactive sub-model variable, augmenting the active sub-model variables.

For the variable mappings, the primary and secondary specifications are lists of strings that are used to target specific sub-model variables and their sub-parameters, respectively. The primary strings are matched to sub-model variable descriptors such as `'cdv_1'` (either user-supplied or default labels). The secondary strings are matched to random variable distribution parameters such as `'mean'` or `'num_trials'` or design/state variable sub-parameters such as `'lower_bound'` or `'upper_bound'`.

An important limitation is that real-valued top-level variables must map to real-valued sub-model variables or real-valued sub-parameters, and integer-valued top-level variables must map to either integer-valued sub-model variables or integer-valued sub-parameters. However, as long as these real versus integer constraints are satisfied, mappings are free to cross variable types (design, aleatory uncertain, epistemic uncertain, state) and domain types (continuous, discrete).

Both `primary_variable_mapping` and `secondary_variable_mapping` specifications are optional, which is designed to support the following three possibilities:

1. If both primary and secondary variable mappings are specified, then an active top-level variable value will be inserted into the identified sub-parameter (the secondary mapping) for the identified sub-model variable (the primary mapping).
2. If a primary mapping is specified but a secondary mapping is not, then an active top-level variable value will be inserted into the identified sub-model variable value (the primary mapping).
3. If a primary mapping is not specified (corresponding secondary mappings, if specified, are ignored), then an active top-level variable value will be inserted into a corresponding sub-model variable, based on matching of variable types (e.g., top-level and sub-model variable specifications both allocate a set of 'continuous_design' variables which are active at the top level). Multiple sub-model variable types may be updated in this manner, provided that they are all active in the top-level variables. Since there is a direct variable correspondence for these default insertions, sub-model bounds and descriptors are also updated from the top-level bounds and labels in order to eliminate the need for redundant input file specifications. Thus, it is typical for the sub-model variables specification to only contain the minimal required information, such as the number of variables of each type, for these insertion targets. The sub-model must allocate enough space for each of the types that will accept default insertions, and the leading set of matching sub-model variables are updated (i.e., the sub-model may allocate more than needed and the trailing set will be unmodified).

These different variable mapping possibilities may be used in any combination by employing empty strings (") for particular omitted mappings (the number of strings in user-supplied primary and secondary variable mapping specifications must equal the total number of active top-level variables, including both continuous and discrete types). The ordering of the active variables is the same as shown in `dakota.input.summary` on [Input Spec Summary](#) and as presented in the [variables](#) section of this manual.

Inactive nested model variables are treated differently from those in the active view. If inactive variables are present at the outer level, then the default type 3 mapping is used for these variables; that is, outer loop inactive variables are inserted into inner loop variables (active or inactive) based on matching of variable types, top-level bounds and labels are also propagated, the inner loop must allocate sufficient space to receive the outer loop values, and the leading subset within this inner loop allocation is updated. This capability is important for allowing nesting beyond two levels, since an active variable at the outer-most loop may become inactive at the next lower level, but still needs to be further propagated down to lower levels in the recursion.

Response Mappings

For the response mappings, the primary and secondary specifications determine how results from the completed sub-iterator are mapped into nested model responses. The response mapping defines a matrix which scales and combines the results from the inner loop into outer loop responses. Each row of the mapping corresponds to one outer loop response, and each column of the mapping corresponds to a result from the inner loop. The results returned from the sub-model are best thought of as forming a column vector; the nested model responses are then the dot-product of the mapping matrix and results vector. The number and type of results that are available and must be accounted for in the response mapping depends on the sub-iterator type:

optimization: the final objective function(s) and nonlinear constraints

nonlinear least squares: the final least squares terms and nonlinear constraints

aleatory uncertainty quantification (UQ): for each response function, a mean statistic, a standard deviation statistic, and all probability/reliability/generalized reliability/response level results for any user-specified `response_levels`, `probability_levels`, `reliability_levels`, and/or `gen_reliability_levels`, in that order.

epistemic and mixed aleatory/epistemic UQ using interval estimation methods: lower and upper interval bounds for each response function.

epistemic and mixed aleatory/epistemic UQ using evidence methods: for each response function, lower and upper interval bounds (belief and plausibility) for all probability/reliability/generalized reliability/response level results computed from any user-specified `response_levels`, `probability_levels`, `reliability_levels`, and/or `gen_reliability_levels`, in that order. parameter studies and design of experiments: for optimization and least squares response data sets, the best solution found (lowest constraint violation if infeasible, lowest composite objective function if feasible). For generic response data sets, a best solution metric is not defined, so the sub-iterator response vector is empty in this case.

The `primary_response_mapping` matrix maps sub-iterator results into top-level objective functions, least squares terms, or generic response functions, depending on the declared top-level response set. The `secondary_response_mapping` matrix maps sub-iterator results into top-level nonlinear inequality and equality constraints. Alternately, if all the responses of the sub-iterator are to be mapped one-to-one to the top-level nested model responses [identity_response_mapping](#) may be specified instead.

Summary

The nested model constructs admit a wide variety of multi-iterator, multi-model solution approaches. For example, optimization within optimization (for hierarchical multidisciplinary optimization), uncertainty quantification within uncertainty quantification (for second-order probability), uncertainty quantification within optimization (for optimization under uncertainty), and optimization within uncertainty quantification (for uncertainty of optima) are all supported, with and without surrogate model indirection. Several examples of nested model usage are provided in the Users Manual, most notably mixed epistemic-aleatory UQ, optimization under uncertainty (OUU), and surrogate-based UQ.

Examples

Two examples are provided to illustrate nested models. The first is a relatively simple case. Although it is somewhat contrived, it demonstrates several features of nested models. A step-by-step explanation is provided below.

```
environment
  method_pointer 'opt'

method
  id_method 'opt'
  asynch_pattern_search
  model_pointer 'outer_model'
  output verbose

model
  id_model 'outer_model'
  variables_pointer 'outer_vars'
  responses_pointer 'outer_resps'
  nested
    sub_method_pointer 'UQ_method'
    primary_variable_mapping 'x1' 'x2'
    secondary_variable_mapping 'mean' 'mean'
    primary_response_mapping 0 0 0.3 0 0 0.7

variables
  id_variables 'outer_vars'
  continuous_design 2
  descriptors 'x1_mean' 'x2_mean'
  lower_bounds -2.0 -2.0
  upper_bounds 2.0 2.0

responses
  id_responses 'outer_resps'
  objective_functions 1
  descriptors 'sum_p'
  no_gradients
  no_hessians

method
  id_method 'UQ_method'
  sampling samples 30
  seed 1234
  model_pointer 'UQ_model'
```

```

probability_levels 0.9 0.9

model
  id_model 'UQ_model'
  single
  variables_pointer 'inner_vars'
  responses_pointer 'inner_resps'
  interface_pointer 'inf'

variables
  id_variables 'inner_vars'
  normal_uncertain 2
  descriptors 'x1' 'x2'
  means 0.0 0.0
  std_deviations 1.0 1.0

responses
  id_responses 'inner_resps'
  response_functions 2
  descriptors 'f1' 'f2'
  no_gradients
  no_hessians

interface
  id_interface 'inf'
  direct
  analysis_drivers 'text_book'

```

The example input is an 'optimization under uncertainty' or OUU. In an OUU, some statistic of the simulation response is optimized. In this case, a weighted sum of the 90th percentiles of two simulation responses from the `text_book` driver is being minimized. The uncertainty in these responses is driven by uncertainty in the input variables, which are normally distributed. The means of the uncertain variables are the design variables in the optimization.

In Dakota, this is accomplished by nesting an uncertainty quantification method ([sampling](#)), which computes the 90th percentiles, within an optimizer (`method-asynch_pattern_search`), which iteratively adjusts the variable means to minimize the objective. For every evaluation requested by the optimizer, the design variables (`x1_mean` and `x2_mean` in the example) are inserted into the means of the uncertain variables (`x1` and `x2`), then the `sampling` study is run to completion, and finally the resulting statistics for the responses `f1` and `f2` are mapped into the responses `sum_p`.

It may be helpful to sketch out the relationships between blocks as indicated by their various pointers in order to understand the nested structure of the study. The top-level or outer method, 'opt', has a model named 'outer_model'. This model refers to variables and responses blocks ('outer_vars' and 'outer_resps') that are used by the optimizer. The 'outer_model' block also refers to a sub-method, 'UQ_method' which defines the UQ study and has its own model, which in turn possess variables, an interface, and responses.

Next, note the primary and secondary variable mappings in 'outer_model', which specify that the values of the active design variables `x1_mean` and `x2_mean` are to be inserted into the means of the uncertain variables `x1` and `x2`.

The response mapping in 'outer_model' is a matrix with a single row and six columns. The single row corresponds to the number of responses in the outer method. As described above, for each of our two sub-model responses, the `sampling` method returns a mean, a standard deviation, and a single probability level in that order to the nested model. There are a total of six results and accordingly six columns in the mapping matrix. The coefficients 0.3 and 0.7 in the matrix result in a weighted sum of the 90th percentiles:

$$sum_p = 0.3 * (90th\ percentile\ of\ f1) + 0.7 * (90th\ percentile\ of\ f2)$$

The next example is a fragment of an input file, showing only the nested model itself. It illustrates more complex variable and response mappings.

```

primary_variable_mapping = ' ' 'X' 'Y'
secondary_variable_mapping = ' ' 'mean' 'mean'
primary_response_mapping = 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
secondary_response_mapping = 0. 0. 0. 1. 3. 0. 0. 0. 0. 0.
                             0. 0. 0. 0. 0. 0. 0. 1. 3. 0.

```

The variable mappings correspond to 4 top-level variables, the first two of which employ the default mappings from active top-level variables to sub-model variables of the same type (option 3 above) and the latter two of which are inserted into the mean distribution parameters of sub-model variables 'X' and 'Y' (option 1 above).

The response mappings define a 3 by 9 matrix corresponding to 9 inner loop results and 3 outer loop response functions (one primary response function and 2 secondary functions, such as one objective and two constraints). Each row of the response mapping is a vector which is multiplied (i.e, with a dot-product) against the 9 sub-iterator results to determine the outer loop function.

Consider again a UQ example with 3 response functions, each providing a mean, a standard deviation, and one level mapping (if no level mappings are specified, the responses would only have a mean and standard deviation). The primary response mapping can be seen to extract the first result from the inner loop, which would correspond to the mean of the first response function. This mapped sub-iterator response becomes a single objective function, least squares term, or generic response function at the outer level, as dictated by the top-level response specification. The secondary response mapping maps the fourth sub-iterator result plus 3 times the fifth sub-iterator result (mean plus 3 standard deviations) into one top-level nonlinear constraint and the seventh sub-iterator result plus 3 times the eighth sub-iterator result (mean plus 3 standard deviations) into another top-level nonlinear constraint, where these top-level nonlinear constraints may be inequality or equality, as dictated by the top-level response specification.

Note that a common case is for each sub-iterator result to be mapped to a unique outer loop response (for example, in the nested UQ case where one wants to determine an interval on each inner loop statistic). In these simple cases, the response mapping would define an identity matrix and [identity_response_mapping](#) may instead be specified.

7.3.4.2.1 `iterator_servers`

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [iterator_servers](#)

Specify the number of iterator servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_servers` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to [ParallelLibrary](#) and the [Parallel Computing](#) chapter of the Users Manual [5] for additional information.

7.3.4.2.2 `iterator_scheduling`

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [iterator_scheduling](#)

Specify the scheduling of concurrent iterators when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Server Mode (Group 1)	Dakota Keyword	Dakota Keyword Description
			master	Specify a dedicated master partition for parallel iterator scheduling
			peer	Specify a peer partition for parallel iterator scheduling

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `iterator_scheduling` specification supports user override of the automatic parallel configuration for the number of iterator servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to ParallelLibrary and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.3.4.2.2.1 `master`

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [iterator_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the iterator servers. This reduces the number of processors available to create servers by 1.

7.3.4.2.2.2 peer

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [iterator_scheduling](#)
- [peer](#)

Specify a peer partition for parallel iterator scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to iterator servers. Note that unlike the case of `evaluation_scheduling`, it is not possible to specify `static` or `dynamic`.

7.3.4.2.3 processors_per_iterator

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [processors_per_iterator](#)

Specify the number of processors per iterator server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Description

An important feature for component-based iterators is that execution of sub-iterator runs may be performed concurrently. The optional `processors_per_iterator` specification supports user override of the automatic parallel configuration for the number of processors in each iterator server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the iterator parallelism level. Currently, `hybrid`, `multi_start`, and `pareto_set` component-based iterators support concurrency in their sub-iterators. Refer to `ParallelLibrary` and the `Parallel Computing` chapter of the `Users Manual [5]` for additional information.

7.3.4.2.4 primary_variable_mapping

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [primary_variable_mapping](#)

Primary mapping of top-level variables to sub-model variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: default variable insertions based on variable type

Description

The `primary_variable_mapping`, `secondary_variable_mapping`, `primary_response_mapping`, and `secondary_response_mapping` keywords control how top-level variables and responses are mapped to variables and responses in the sub-model. Their usage is explained on the parent keyword (`sub_method_pointer`) page.

7.3.4.2.5 `secondary_variable_mapping`

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [secondary_variable_mapping](#)

Secondary mapping of top-level variables to sub-model variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: primary mappings into sub-model variables are value-based

Description

The `primary_variable_mapping`, `secondary_variable_mapping`, `primary_response_mapping`, and `secondary_response_mapping` keywords control how top-level variables and responses are mapped to variables and responses in the sub-model. Their usage is explained on the parent keyword (`sub_method_pointer`) page.

7.3.4.2.6 `primary_response_mapping`

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [primary_response_mapping](#)

Mapping of sub-method results to top-level primary responses

Specification

Alias: none

Argument(s): REALLIST

Default: no sub-iterator contribution to primary functions

Description

Matrix that specifies how sub-method results (statistics, best parameters, etc.) map to top-level nested model primary responses (objectives, calibration terms, or response functions). Its usage is explained in detail on the parent keyword (`sub_method_pointer`) page.

7.3.4.2.7 `secondary_response_mapping`

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [secondary_response_mapping](#)

Mapping of sub-method results to top-level secondary responses

Specification

Alias: none

Argument(s): REALLIST

Default: no sub-iterator contribution to secondary functions

Description

Matrix that specifies how sub-method results (statistics, best parameters, etc.) map to top-level nested model secondary responses (nonlinear inequality or equality constraints). Its usage is explained in detail on the parent keyword (`sub_method_pointer`) page.

7.3.4.2.8 `identity_response_mapping`

- [Keywords Area](#)
- [model](#)
- [nested](#)
- [sub_method_pointer](#)
- [identity_response_mapping](#)

Identity mapping of sub-method results to top-level responses

Specification

Alias: none

Argument(s): none

Default: no sub-iterator contribution to nested model functions

Description

Specifies that the list of sub-method results (statistics, best parameters, etc.) should map one-to-one to the list of top-level nested model responses.

Usage Tips

To use an identity mapping, the number of top-level nested model responses (primary + secondary responses) must equal the number of sub-method results. If receiving an error about this, set `output verbose` and re-run the Dakota study to see a list of sub-method results responses.

The identity map may not be used in conjunction with [optional_interface_pointer](#), nor with an explicit `primary_response_mapping` or `secondary_response_mapping`.

Examples

```
model
  id_model 'nested'
  nested
  sub_method_pointer 'aleat'
  identity_response_mapping
```

7.3.4.2.9 Input Spec Summary

This file is derived automatically from `dakota.xml`, which is used in the generation of parser system files that are compiled into the Dakota executable. Therefore, these files are the definitive source for input syntax, capability options, and associated data inputs. Refer to the Developers Manual information on how to modify the input specification and propagate the changes through the parsing system.

Key features of the input specification and the associated user input files include:

- In the input specification, required individual specifications simply appear, optional individual and group specifications are enclosed in `[]`, required group specifications are enclosed in `()`, and either-or relationships are denoted by the `|` symbol. These symbols only appear in `dakota.input.summary`; they must not appear in actual user input files.
- Keyword specifications (i.e., `environment`, `method`, `model`, `variables`, `interface`, and `responses`) begin with the keyword possibly preceded by white space (blanks, tabs, and newlines) both in the input specifications and in user input files. For readability, keyword specifications may be spread across several lines. Earlier versions of Dakota (prior to 4.1) required a backslash character (`\`) at the ends of intermediate lines of a keyword. While such backslashes are still accepted, they are no longer required.
- Some of the keyword components within the input specification indicate that the user must supply `INTEGER`, `REAL`, `STRING`, `INTEGERLIST`, `REALLIST`, or `STRINGLIST` data as part of the specification. In a user input file, the `"="` is optional, data in a `LIST` can be separated by commas or whitespace, and the `STRING` data are enclosed in single or double quotes (e.g., `'text_book'` or `"text_book"`).
- In user input files, input is largely order-independent (except for entries in lists of data), case insensitive, and white-space insensitive. Although the order of input shown in the [Sample Input Files](#) generally follows the order of options in the input specification, this is not required.
- In user input files, specifications may be abbreviated so long as the abbreviation is unique. For example, the `npsol_sqp` specification within the `method` keyword could be abbreviated as `npsol`, but `dot_sqp` should not be abbreviated as `dot` since this would be ambiguous with other `DOT` method specifications.
- In both the input specification and user input files, comments are preceded by `#`.
- `ALIAS` refers to synonymous keywords, which often exist for backwards compatibility. Users are encouraged to use the most current keyword.

dakota.input.summary:

```

KEYWORD01 environment
  [ tabular_data ALIAS tabular_graphics_data
    [ tabular_data_file ALIAS tabular_graphics_file STRING ]
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
      | annotated
      | freeform ]
    ]
  [ output_file STRING ]
  [ error_file STRING ]
  [ read_restart STRING
    [ stop_restart INTEGER >= 0 ]
  ]
  [ write_restart STRING ]
  [ output_precision INTEGER >= 0 ]
  [ results_output
    [ results_output_file STRING ]
    [ text ]
    [ hdf5
      [ model_selection
        top_method
        | none
        | all_methods
        | all
        ]
      [ interface_selection
        none
        | simulation
        | all
        ]
      ]
    ]
  ]
  [ graphics ]
  [ check ]
  [ pre_run
    [ input STRING ]
    [ output STRING
      [ ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
        )
        | annotated
        | freeform ]
      ]
    ]
  ]
  [ run
    [ input STRING ]
    [ output STRING ]
  ]
  [ post_run
    [ input STRING
      [ ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
        )
        | annotated
        | freeform ]
      ]
    [ output STRING ]
  ]
  [ top_method_pointer ALIAS method_pointer STRING ]

KEYWORD method
  [ id_method STRING ]
  [ output
    debug
  ]

```

```

| verbose
| normal
| quiet
| silent
]
[ final_solutions INTEGER >= 0 ]
( hybrid
  ( sequential ALIAS uncoupled
    ( method_name_list STRINGLIST
      [ model_pointer_list STRINGLIST ]
    )
    | method_pointer_list STRINGLIST
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
  )
  |
  ( embedded ALIAS coupled
    ( global_method_name STRING
      [ global_model_pointer STRING ]
    )
    | global_method_pointer STRING
    ( local_method_name STRING
      [ local_model_pointer STRING ]
    )
    | local_method_pointer STRING
    [ local_search_probability REAL ]
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
  )
  |
  ( collaborative
    ( method_name_list STRINGLIST
      [ model_pointer_list STRINGLIST ]
    )
    | method_pointer_list STRINGLIST
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
  )
  |
  ( multi_start
    ( method_name STRING
      [ model_pointer STRING ]
    )
    | method_pointer STRING
    [ random_starts INTEGER
      [ seed INTEGER ]
    ]
    [ starting_points REALLIST ]
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
  )
  |
  ( pareto_set
    ( method_name ALIAS opt_method_name STRING
      [ model_pointer ALIAS opt_model_pointer STRING ]
    )
  )

```

```

| method_pointer ALIAS opt_method_pointer STRING
[ random_weight_sets INTEGER
  [ seed INTEGER ]
]
[ weight_sets ALIAS multi_objective_weight_sets REALLIST ]
[ iterator_servers INTEGER > 0 ]
[ iterator_scheduling
  master
  | peer
]
[ processors_per_iterator INTEGER > 0 ]
)
|
( branch_and_bound
method_pointer STRING
|
( method_name STRING
  [ model_pointer STRING ]
)
[ scaling ]
)
|
( surrogate_based_local
method_pointer ALIAS approx_method_pointer STRING
| method_name ALIAS approx_method_name STRING
model_pointer ALIAS approx_model_pointer STRING
[ soft_convergence_limit INTEGER ]
[ truth_surrogate_bypass ]
[ approx_subproblem
  original_primary
  | single_objective
  | augmented_lagrangian_objective
  | lagrangian_objective
original_constraints
| linearized_constraints
| no_constraints
]
[ merit_function
  penalty_merit
  | adaptive_penalty_merit
  | lagrangian_merit
  | augmented_lagrangian_merit
]
[ acceptance_logic
  tr_ratio
  | filter
]
[ constraint_relax
  homotopy
]
[ trust_region
  [ initial_size REALLIST ]
  [ minimum_size REAL ]
  [ contract_threshold REAL ]
  [ expand_threshold REAL ]
  [ contraction_factor REAL ]
  [ expansion_factor REAL ]
]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ constraint_tolerance REAL ]
)
|
( surrogate_based_global
method_pointer ALIAS approx_method_pointer STRING
| method_name ALIAS approx_method_name STRING
model_pointer ALIAS approx_model_pointer STRING
[ replace_points ]
[ max_iterations INTEGER >= 0 ]
)
|
( dot_frcg
  [ max_iterations INTEGER >= 0 ]
)

```



```

    [ convergence_tolerance REAL ]
    [ constraint_tolerance REAL ]
    [ speculative ]
    [ max_function_evaluations INTEGER >= 0 ]
    [ scaling ]
    [ model_pointer STRING ]
  )
|
( dot_mmfd
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( dot_bfgs
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( dot_slp
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( dot_sqp
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( conmin_frcg
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( conmin_mfd
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( dl_solver STRING
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)

```

```

|
( npsol_sqp
  [ verify_level INTEGER ]
  [ function_precision REAL ]
  [ linesearch_tolerance REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nlssol_sqp
  [ verify_level INTEGER ]
  [ function_precision REAL ]
  [ linesearch_tolerance REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ constraint_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nlpql_sqp
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_cg
  [ max_step REAL ]
  [ gradient_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_q_newton
  [ search_method
    value_based_line_search
    | gradient_based_line_search
    | trust_region
    | tr_pds
  ]
  [ merit_function
    el_bakry
    | argaez_tapia
    | van_shanno
  ]
  [ steplength_to_boundary REAL ]
  [ centering_parameter REAL ]
  [ max_step REAL ]
  [ gradient_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_fd_newton
  [ search_method
    value_based_line_search

```

```

    | gradient_based_line_search
    | trust_region
    | tr_pds
  ]
[ merit_function
  el_bakry
  | argaez_tapia
  | van_shanno
]
[ steplength_to_boundary REAL ]
[ centering_parameter REAL ]
[ max_step REAL ]
[ gradient_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ speculative ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( optpp_g_newton
  [ search_method
    value_based_line_search
    | gradient_based_line_search
    | trust_region
    | tr_pds
  ]
  [ merit_function
    el_bakry
    | argaez_tapia
    | van_shanno
  ]
  [ steplength_to_boundary REAL ]
  [ centering_parameter REAL ]
  [ max_step REAL ]
  [ gradient_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_newton
  [ search_method
    value_based_line_search
    | gradient_based_line_search
    | trust_region
    | tr_pds
  ]
  [ merit_function
    el_bakry
    | argaez_tapia
    | van_shanno
  ]
  [ steplength_to_boundary REAL ]
  [ centering_parameter REAL ]
  [ max_step REAL ]
  [ gradient_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( optpp_pds
  [ search_scheme_size INTEGER ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]

```

```

    [ max_function_evaluations INTEGER >= 0 ]
    [ scaling ]
    [ model_pointer STRING ]
  )
|
( demo_tpl
  [ max_function_evaluations INTEGER >= 0 ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ options_file STRING ]
)
|
( rol
  [ max_iterations INTEGER >= 0 ]
  [ variable_tolerance REAL ]
  [ gradient_tolerance REAL ]
  [ constraint_tolerance REAL ]
  [ options_file STRING ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( asynch_pattern_search ALIAS coliny_apps
  [ initial_delta REAL ]
  [ contraction_factor REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ synchronization
    blocking
    | nonblocking
  ]
  [ merit_function
    merit_max
    | merit_max_smooth
    | merit1
    | merit1_smooth
    | merit2
    | merit2_smooth
    | merit2_squared
  ]
  [ constraint_penalty REAL ]
  [ smoothing_factor REAL ]
  [ constraint_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( mesh_adaptive_search
  [ initial_delta REAL ]
  [ variable_tolerance REAL ]
  [ function_precision REAL ]
  [ seed INTEGER > 0 ]
  [ history_file STRING ]
  [ display_format STRING ]
  [ variable_neighborhood_search REAL ]
  [ neighbor_order INTEGER > 0 ]
  [ display_all_evaluations ]
  [ use_surrogate
    inform_search
    | optimize
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nowpac
  [ trust_region
    [ initial_size REALLIST ]
  ]

```

```

    [ minimum_size REAL ]
    [ contract_threshold REAL ]
    [ expand_threshold REAL ]
    [ contraction_factor REAL ]
    [ expansion_factor REAL ]
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( snowpac
  [ seed INTEGER > 0 ]
  [ trust_region
    [ initial_size REALLIST ]
    [ minimum_size REAL ]
    [ contract_threshold REAL ]
    [ expand_threshold REAL ]
    [ contraction_factor REAL ]
    [ expansion_factor REAL ]
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( moga
  [ fitness_type
    layer_rank
    | domination_count
  ]
  [ replacement_type
    elitist
    | roulette_wheel
    | unique_roulette_wheel
    |
    ( below_limit REAL
      [ shrinkage_fraction ALIAS shrinkage_percentage REAL ]
    )
  ]
  [ niching_type
    radial REALLIST
    | distance REALLIST
    |
    ( max_designs REALLIST
      [ num_designs INTEGER >= 2 ]
    )
  ]
  [ convergence_type
    metric_tracker
    [ percent_change REAL ]
    [ num_generations INTEGER >= 0 ]
  ]
  [ postprocessor_type
    orthogonal_distance REALLIST
  ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ population_size INTEGER >= 0 ]
  [ log_file STRING ]
  [ print_each_pop ]
  [ initialization_type
    simple_random
    | unique_random
    | flat_file STRING
  ]
  [ crossover_type
    multi_point_binary INTEGER
    | multi_point_parameterized_binary INTEGER
    | multi_point_real INTEGER
  ]
)

```

```

|
( shuffle_random
  [ num_parents INTEGER > 0 ]
  [ num_offspring INTEGER > 0 ]
)
[ crossover_rate REAL ]
]
[ mutation_type
bit_random
| replace_uniform
|
( offset_normal
  [ mutation_scale REAL ]
)
|
( offset_cauchy
  [ mutation_scale REAL ]
)
|
( offset_uniform
  [ mutation_scale REAL ]
)
[ mutation_rate REAL ]
]
[ seed INTEGER > 0 ]
[ convergence_tolerance REAL ]
[ model_pointer STRING ]
)
|
( soja
[ fitness_type
merit_function
  [ constraint_penalty REAL ]
]
[ replacement_type
elitist
| favor_feasible
| roulette_wheel
| unique_roulette_wheel
]
[ convergence_type
( best_fitness_tracker
  [ percent_change REAL ]
  [ num_generations INTEGER >= 0 ]
)
|
( average_fitness_tracker
  [ percent_change REAL ]
  [ num_generations INTEGER >= 0 ]
)
]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ population_size INTEGER >= 0 ]
[ log_file STRING ]
[ print_each_pop ]
[ initialization_type
simple_random
| unique_random
| flat_file STRING
]
[ crossover_type
multi_point_binary INTEGER
| multi_point_parameterized_binary INTEGER
| multi_point_real INTEGER
|
( shuffle_random
  [ num_parents INTEGER > 0 ]
  [ num_offspring INTEGER > 0 ]
)
[ crossover_rate REAL ]
]

```

```

[ mutation_type
  bit_random
  | replace_uniform
  |
  ( offset_normal
    [ mutation_scale REAL ]
  )
  |
  ( offset_cauchy
    [ mutation_scale REAL ]
  )
  |
  ( offset_uniform
    [ mutation_scale REAL ]
  )
  [ mutation_rate REAL ]
]
[ seed INTEGER > 0 ]
[ convergence_tolerance REAL ]
[ model_pointer STRING ]
)
|
( coliny_pattern_search
  [ constant_penalty ]
  [ no_expansion ]
  [ expand_after_success INTEGER ]
  [ pattern_basis
    coordinate
    | simplex
  ]
  [ stochastic ]
  [ total_pattern_size INTEGER ]
  [ exploratory_moves
    multi_step
    | adaptive_pattern
    | basic_pattern
  ]
  [ synchronization
    blocking
    | nonblocking
  ]
  [ contraction_factor REAL ]
  [ constraint_penalty REAL ]
  [ initial_delta REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( coliny_solis_wets
  [ contract_after_failure INTEGER ]
  [ no_expansion ]
  [ expand_after_success INTEGER ]
  [ constant_penalty ]
  [ contraction_factor REAL ]
  [ constraint_penalty REAL ]
  [ initial_delta REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
)

```

```

    [ model_pointer STRING ]
  )
|
( coliny_cobyla
  [ initial_delta REAL ]
  [ variable_tolerance REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( coliny_direct
  [ division
    major_dimension
    | all_dimensions
  ]
  [ global_balance_parameter REAL ]
  [ local_balance_parameter REAL ]
  [ max_boxsize_limit REAL ]
  [ min_boxsize_limit REAL ]
  [ constraint_penalty REAL ]
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( coliny_ea
  [ population_size INTEGER > 0 ]
  [ initialization_type
    simple_random
    | unique_random
    | flat_file STRING
  ]
  [ fitness_type
    linear_rank
    | merit_function
  ]
  [ replacement_type
    random INTEGER
    | chc INTEGER
    | elitist INTEGER
    [ new_solutions_generated INTEGER ]
  ]
  [ crossover_rate REAL ]
  [ crossover_type
    two_point
    | blend
    | uniform
  ]
  [ mutation_rate REAL ]
  [ mutation_type
    replace_uniform
    |
    ( offset_normal
      [ mutation_scale REAL ]
      [ mutation_range INTEGER ]
    )
    |
    ( offset_cauchy
      [ mutation_scale REAL ]
      [ mutation_range INTEGER ]
    )
  ]
)

```



```

    )
  |
  ( offset_uniform
    [ mutation_scale REAL ]
    [ mutation_range INTEGER ]
  )
  [ non_adaptive ]
]
[ constraint_penalty REAL ]
[ solution_target ALIAS solution_accuracy REAL ]
[ seed INTEGER > 0 ]
[ show_misc_options ]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ scaling ]
[ model_pointer STRING ]
)
|
( coliny_beta
  beta_solver_name STRING
  [ solution_target ALIAS solution_accuracy REAL ]
  [ seed INTEGER > 0 ]
  [ show_misc_options ]
  [ misc_options STRINGLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nl2sol
  [ function_precision REAL ]
  [ absolute_conv_tol REAL ]
  [ x_conv_tol REAL ]
  [ singular_conv_tol REAL ]
  [ singular_radius REAL ]
  [ false_conv_tol REAL ]
  [ initial_trust_radius REAL ]
  [ covariance INTEGER ]
  [ regression_diagnostics ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ speculative ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( nonlinear_cg
  [ misc_options STRINGLIST ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( ncsu_direct
  [ solution_target ALIAS solution_accuracy REAL ]
  [ min_boxsize_limit REAL ]
  [ volume_boxsize_limit REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( genie_opt_darts
  [ seed INTEGER > 0 ]
  [ max_function_evaluations INTEGER >= 0 ]

```

```

    [ scaling ]
    [ model_pointer STRING ]
  )
|
( genie_direct
  [ seed INTEGER > 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ scaling ]
  [ model_pointer STRING ]
)
|
( efficient_global
  [ initial_samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ batch_size INTEGER >= 1
    [ exploration INTEGER >= 0 ]
    [ synchronization
      blocking
      | nonblocking
    ]
  ]
  [ max_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ x_conv_tol REAL ]
  [ gaussian_process ALIAS kriging
    ( surfpack
      [ export_model
      [ filename_prefix STRING ]
      (formats
        [ text_archive ]
        [ binary_archive ]
      )
    )
  ]
  | dakota
  |
  ( experimental
    [ export_model
    [ filename_prefix STRING ]
    ( formats
      [ text_archive ]
      [ binary_archive ]
    )
  ]
  [ options_file STRING ]
)
]
[ use_derivatives ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ model_pointer STRING ]
)
|
( surrogate_based_uq
  [ samples_on_emulator ALIAS samples INTEGER ]
  [ sample_type

```

```

    lhs
    | random
    ]
  [ rng
    mt19937
    | rnum2
    ]
  [ probability_refinement ALIAS sample_refinement
    import
    | adapt_import
    | mm_adapt_import
    [ refinement_samples INTEGERLIST ]
    ]
  [ final_moments
    none
    | standard
    | central
    ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | reliabilities
      | gen_reliabilities
      [ system
        series
        | parallel
        ]
      ]
    ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ]
    ]
  [ reliability_levels REALLIST
    [ num_reliability_levels INTEGERLIST ]
    ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ]
    ]
  [ distribution
    cumulative
    | complementary
    ]
  [ variance_based_decomp
    [ interaction_order INTEGER > 0 ]
    [ drop_tolerance REAL ]
    ]
  [ diagonal_covariance
  | full_covariance ]
  [ import_approx_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
    | annotated
    | freeform ]
    [ active_only ]
    ]
  [ export_approx_points_file ALIAS export_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
    | annotated
    | freeform ]
    ]
  [ seed INTEGER > 0 ]
  [ fixed_seed ]
  [ model_pointer STRING ]
  )
|

```

```

( function_train
  [ p_refinement
    ( uniform
      increment_start_rank
      | increment_start_order
      | increment_max_rank
      | increment_max_order
      | increment_max_rank_order
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ regression_type
    ls
    |
    ( rls2
      l2_penalty REAL
    )
  ]
  [ max_solver_iterations INTEGER >= 0 ]
  [ max_cross_iterations INTEGER >= 0 ]
  [ solver_tolerance REAL ]
  [ response_scaling ]
  [ tensor_grid ]
  collocation_points INTEGER
  | collocation_ratio REAL
  [ rounding_tolerance REAL ]
  [ arithmetic_tolerance REAL ]
  [ start_order ALIAS order INTEGER >= 0
    [ dimension_preference REALLIST ]
  ]
  [ adapt_order ]
  [ kick_order INTEGER > 0 ]
  [ max_order INTEGER >= 0 ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  [ start_rank ALIAS rank INTEGER >= 0 ]
  [ adapt_rank ]
  [ kick_rank INTEGER > 0 ]
  [ max_rank INTEGER >= 0 ]
  [ max_cv_rank_candidates INTEGER >= 0 ]
  [ samples_on_emulator ALIAS samples INTEGER ]
  [ sample_type
    lhs
    | random
  ]
  [ rng
    mt19937
    | rnum2
  ]
  [ probability_refinement ALIAS sample_refinement
    import
    | adapt_import
    | mm_adapt_import
    [ refinement_samples INTEGERLIST ]
  ]
  [ final_moments
    none
    | standard
    | central
  ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | reliabilities
      | gen_reliabilities
      [ system
        series
        | parallel
      ]
    ]
  ]

```

```

]
]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ seed INTEGER > 0 ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( multifidelity_function_train
  [ p_refinement
    ( uniform
      increment_start_rank
      | increment_start_order
      | increment_max_rank
      | increment_max_order
      | increment_max_rank_order
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ statistics_mode
    active
    | combined
  ]
  [ allocation_control
    greedy
  ]
  [ discrepancy_emulation
    distinct ALIAS paired
    | recursive
  ]
]

```

```

[ rounding_tolerance REAL ]
[ arithmetic_tolerance REAL ]
[ regression_type
  ls
  |
  ( rls2
    l2_penalty REAL
  )
]
[ max_solver_iterations INTEGER >= 0 ]
[ max_cross_iterations INTEGER >= 0 ]
[ solver_tolerance REAL ]
[ response_scaling ]
[ tensor_grid ]
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ collocation_ratio REAL ]
[ start_order_sequence ALIAS order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
]
[ adapt_order ]
[ kick_order INTEGER > 0 ]
[ max_order INTEGER >= 0 ]
[ max_cv_order_candidates INTEGER >= 0 ]
[ start_rank_sequence ALIAS rank_sequence INTEGERLIST ]
[ adapt_rank ]
[ kick_rank INTEGER > 0 ]
[ max_rank INTEGER >= 0 ]
[ max_cv_rank_candidates INTEGER >= 0 ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
]
[ rng
  mt19937
  | rnum2
]
[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
]
[ final_moments
  none
  | standard
  | central
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]

```

```

[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( multilevel_function_train
  [ max_iterations INTEGER >= 0 ]
  [ allocation_control
    ( estimator_variance
      [ estimator_rate REAL ]
    )
    | rank_sampling
  ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ discrepancy_emulation
    distinct ALIAS paired
    | recursive
  ]
  [ rounding_tolerance REAL ]
  [ arithmetic_tolerance REAL ]
  [ regression_type
    ls
    |
    ( rls2
      l2_penalty REAL
    )
  ]
  [ max_solver_iterations INTEGER >= 0 ]
  [ max_cross_iterations INTEGER >= 0 ]
  [ solver_tolerance REAL ]
  [ response_scaling ]
  [ tensor_grid ]
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ collocation_ratio REAL ]
  [ start_order_sequence ALIAS order_sequence INTEGERLIST
    [ dimension_preference REALLIST ]
  ]
  [ adapt_order ]
  [ kick_order INTEGER > 0 ]
  [ max_order INTEGER >= 0 ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  [ start_rank_sequence ALIAS rank_sequence INTEGERLIST ]
  [ adapt_rank ]
  [ kick_rank INTEGER > 0 ]

```

```

[ max_rank INTEGER >= 0 ]
[ max_cv_rank_candidates INTEGER >= 0 ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
  ]
[ rng
  mt19937
  | rnum2
  ]
[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
  ]
[ final_moments
  none
  | standard
  | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
      ]
    ]
  ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ distribution
  cumulative
  | complementary
  ]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
  ]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  ]
[ seed_sequence INTEGERLIST ]

```



```

    [ fixed_seed ]
    [ model_pointer STRING ]
  )
|
( polynomial_chaos ALIAS nond_polynomial_chaos
  [ p_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | decay
      | generalized
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  ( quadrature_order INTEGER
    [ dimension_preference REALLIST ]
    [ nested
    | non_nested ]
  )
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
|
cubature_integrand INTEGER
|
( expansion_order INTEGER
  [ dimension_preference REALLIST ]
  [ basis_type
    tensor_product
    | total_order
  ]
  ( adapted
  [ advancements INTEGER ]
  [ soft_convergence_limit INTEGER ]
)
]
( collocation_points INTEGER
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
)
]
|
  basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
]
|
  ( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
)
]
|
  ( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
)
]
  [ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
]

```

```

    [ ratio_order REAL ]
    [ response_scaling ]
    [ use_derivatives ]
    [ tensor_grid ]
    [ reuse_points ALIAS reuse_samples ]
    [ max_solver_iterations INTEGER >= 0 ]
  )
|
( collocation_ratio REAL
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
  ]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
  ]
  |
  ( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
  ]
  |
  ( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
  ]
  [ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
  )
|
( expansion_samples INTEGER
  [ reuse_points ALIAS reuse_samples ]
  )
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points INTEGER
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
  )
)

```

```

| import_expansion_file STRING
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
  ]
[ rng
  mt19937
  | rnum2
  ]
[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
  ]
[ final_moments
  none
  | standard
  | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
      ]
    ]
  ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ distribution
  cumulative
  | complementary
  ]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
  ]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated

```

```

    | freeform ]
  ]
  [ seed INTEGER > 0 ]
  [ fixed_seed ]
  [ model_pointer STRING ]
)
|
( multifidelity_polynomial_chaos
  [ p_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | decay
      | generalized
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ statistics_mode
    active
    | combined
  ]
  [ allocation_control
    greedy
  ]
  [ discrepancy_emulation
    distinct ALIAS paired
    | recursive
  ]
  ( quadrature_order_sequence INTEGERLIST
    [ dimension_preference REALLIST ]
    [ nested
      | non_nested ]
  )
  |
  ( sparse_grid_level_sequence INTEGERLIST
    [ dimension_preference REALLIST ]
    [ restricted
      | unrestricted ]
    [ nested
      | non_nested ]
  )
  |
  ( expansion_order_sequence INTEGERLIST
    [ dimension_preference REALLIST ]
    [ basis_type
      tensor_product
      | total_order
    ]
    ( adapted
      [ advancements INTEGER ]
      [ soft_convergence_limit INTEGER ]
    )
  ]
  ( collocation_ratio REAL
    [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
    [ ( least_squares
      [ svd
        | equality_constrained ]
      )
    ]
    |
    ( orthogonal_matching_pursuit ALIAS omp
      [ noise_tolerance REALLIST ]
    )
    |
    ( basis_pursuit ALIAS bp
      |
      ( basis_pursuit_denoising ALIAS bpdn
        [ noise_tolerance REALLIST ]
      )
    )
  ]
)

```

```

]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
[ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
[ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
[ header ]
[ eval_id ]
[ interface_id ]
)
| annotated
| freeform ]
[ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
collocation_points_sequence INTEGERLIST
[ tensor_grid INTEGERLIST ]
[ reuse_points ALIAS reuse_samples ]
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
[ header ]
[ eval_id ]
[ interface_id ]
)
| annotated
| freeform ]
[ active_only ]
]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
lhs
| random
]
[ rng
mt19937
| rnum2
]
[ probability_refinement ALIAS sample_refinement
import
| adapt_import
| mm_adapt_import
[ refinement_samples INTEGERLIST ]
]
[ final_moments
none

```

```

    | standard
    | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
  series
  | parallel
  ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ distribution
  cumulative
  | complementary
  ]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
  ]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  ]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( multilevel_polynomial_chaos
  [ max_iterations INTEGER >= 0 ]
  [ allocation_control
    ( estimator_variance
      [ estimator_rate REAL ]
    )
  | rip_sampling
  ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
  ]
  [ discrepancy_emulation
    distinct ALIAS paired

```

```

| recursive
]
( expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
  tensor_product
  | total_order
  |
  ( adapted
[ advancements INTEGER ]
[ soft_convergence_limit INTEGER ]
)
]
( collocation_ratio REAL
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ ( least_squares
[ svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
|
[ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
[ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
[ header ]
[ eval_id ]
[ interface_id ]
)
| annotated
| freeform ]
[ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
collocation_points_sequence INTEGERLIST
[ tensor_grid INTEGERLIST ]
[ reuse_points ALIAS reuse_samples ]
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
[ header ]
[ eval_id ]
[ interface_id ]
]

```

```

    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
  ]
[ rng
  mt19937
  | rnum2
  ]
[ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
  ]
[ final_moments
  none
  | standard
  | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
      ]
    ]
  ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ distribution
  cumulative
  | complementary
  ]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
  ]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated

```



```

    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
    | annotated
    | freeform ]
]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( stoch_collocation ALIAS nond_stoch_collocation
  [ ( p_refinement
      uniform
      |
      ( dimension_adaptive
        sobol
        | generalized
        )
      )
  |
  ( h_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | generalized
      )
    | local_adaptive
    ]
  [ max_refinement_iterations INTEGER >= 0 ]
  [ convergence_tolerance REAL ]
  [ metric_scale
    relative
    | absolute
    ]
  ( quadrature_order INTEGER
    [ dimension_preference REALLIST ]
    [ nested
    | non_nested ]
    )
  |
  ( sparse_grid_level INTEGER
    [ dimension_preference REALLIST ]
    [ nodal
    | hierarchical ]
    [ restricted
    | unrestricted ]
    [ nested
    | non_nested ]
    )
  [ piecewise
  | askey
  | wiener ]
  [ use_derivatives ]
  [ samples_on_emulator ALIAS samples INTEGER ]
  [ sample_type
    lhs
    | random
    ]
  ]
  [ rng
    mt19937
    | rnum2
    ]
  [ probability_refinement ALIAS sample_refinement
    import
    | adapt_import
    | mm_adapt_import
    [ refinement_samples INTEGERLIST ]
    ]
  [ final_moments
    none

```

```

    | standard
    | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
  series
  | parallel
  ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
  ]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
[ distribution
  cumulative
  | complementary
  ]
[ variance_based_decomp
  [ interaction_order INTEGER > 0 ]
  [ drop_tolerance REAL ]
  ]
[ diagonal_covariance
| full_covariance ]
[ import_approx_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  ]
[ seed INTEGER > 0 ]
[ fixed_seed ]
[ model_pointer STRING ]
)
|
( multifidelity_stoch_collocation
  [ ( p_refinement
    uniform
    |
    ( dimension_adaptive
  sobol
  | generalized
  )
  )
  ]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
  sobol

```

```

    | generalized
    )
  | local_adaptive
  ]
[ max_refinement_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ metric_scale
  relative
  | absolute
  ]
[ statistics_mode
  active
  | combined
  ]
[ allocation_control
  greedy
  ]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
  ]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
  )
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
  )
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ samples_on_emulator ALIAS samples INTEGER ]
[ sample_type
  lhs
  | random
  ]
[ rng
  mt19937
  | rnum2
  ]
[ probability_refinement ALIAS sample_refinement
import
  | adapt_import
  | mm_adapt_import
  [ refinement_samples INTEGERLIST ]
  ]
[ final_moments
  none
  | standard
  | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
  probabilities
  | reliabilities
  | gen_reliabilities
  [ system
  series
  | parallel
  ]
  ]
  ]
[ probability_levels REALLIST

```

```

    [ num_probability_levels INTEGERLIST ]
  ]
  [ reliability_levels REALLIST
    [ num_reliability_levels INTEGERLIST ]
  ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ variance_based_decomp
    [ interaction_order INTEGER > 0 ]
    [ drop_tolerance REAL ]
  ]
  [ diagonal_covariance
  | full_covariance ]
  [ import_approx_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
  [ export_approx_points_file ALIAS export_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ seed_sequence INTEGERLIST ]
  [ fixed_seed ]
  [ model_pointer STRING ]
)
|
( sampling ALIAS nond_sampling
  [ samples ALIAS initial_samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ fixed_seed ]
  [ sample_type
    lhs
    | random
    | incremental_lhs
    | incremental_random
  ]
  [ refinement_samples INTEGERLIST ]
  [ d_optimal
    [ candidate_designs INTEGER > 0 ]
    | leja_oversample_ratio REAL ]
  ]
  [ variance_based_decomp
    [ drop_tolerance REAL ]
  ]
  [ backfill ]
  [ principal_components
    [ percent_variance_explained REAL ]
  ]
  [ wilks
    [ order INTEGER ]
    [ confidence_level REAL ]
    [ one_sided_lower ]
    [ one_sided_upper ]
    [ two_sided ]
  ]
  [ final_moments
    none

```

```

    | standard
    | central
  ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( multilevel_sampling ALIAS multilevel_mc ALIAS mlmc
  [ seed_sequence INTEGERLIST ]
  [ fixed_seed ]
  [ pilot_samples ALIAS initial_samples INTEGERLIST ]
  [ solution_mode
    online_pilot
    | offline_pilot
    | pilot_projection
  ]
  [ sample_type
    lhs
    | random
  ]
  [ export_sample_sequence
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ allocation_target
    mean
    |
    ( variance
      [ optimization ]
    )
    |
    ( standard_deviation
      [ optimization ]
    )
    |
    ( scalarization
      [ scalarization_response_mapping REALLIST ]
      [ optimization ]
    )
  ]
]

```

```

[ qoi_aggregation
  sum
  | max
]
[ convergence_tolerance REAL ]
[ convergence_tolerance_type
  relative
  | absolute
]
[ convergence_tolerance_target
  variance_constraint
  | cost_constraint
]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ final_statistics
  estimator_performance
  |
  ( qoi_statistics
    [ final_moments
  none
  | standard
  | central
  ]
    [ distribution
  cumulative
  | complementary
  ]
  )
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( multifidelity_sampling ALIAS multifidelity_mc ALIAS mfmc
  [ seed_sequence INTEGERLIST ]
  [ fixed_seed ]
  [ pilot_samples ALIAS initial_samples INTEGERLIST ]
  [ solution_mode
  online_pilot
  | offline_pilot
  | pilot_projection
  ]
  [ numerical_solve
  [ fallback
  | override ]
  [ sqp
  | nip ]
  ]
  [ sample_type
  lhs
  | random
  ]
  [ export_sample_sequence
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ max_function_evaluations INTEGER >= 0 ]
  [ final_statistics
  estimator_performance
  |
  ( qoi_statistics
    [ final_moments

```

```

    none
    | standard
    | central
  ]
  [ distribution
cumulative
  | complementary
  ]
  )
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( multilevel_multifidelity_sampling ALIAS multilevel_multifidelity_mc ALIAS mlfmc
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ pilot_samples ALIAS initial_samples INTEGERLIST ]
[ solution_mode
  online_pilot
  | offline_pilot
  | pilot_projection
]
[ sample_type
  lhs
  | random
]
[ export_sample_sequence
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ convergence_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ final_statistics
  estimator_performance
  |
  ( goi_statistics
    [ final_moments
  none
  | standard
  | central
  ]
  [ distribution
cumulative
  | complementary
  ]
  )
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( approximate_control_variate ALIAS acv_sampling
acv_independent_sampling ALIAS acv_is
| acv_multifidelity ALIAS acv_mf
| acv_kl ALIAS acv_adaptive
[ pilot_samples ALIAS initial_samples INTEGERLIST ]
[ solution_mode
  online_pilot
  | offline_pilot
  | pilot_projection
]

```

```

]
[ truth_fixed_by_pilot ]
[ sqp
| nip ]
[ seed_sequence INTEGERLIST ]
[ fixed_seed ]
[ sample_type
  lhs
  | random
]
[ export_sample_sequence
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ convergence_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ max_function_evaluations INTEGER >= 0 ]
[ final_statistics
  estimator_performance
  |
  ( qoi_statistics
    [ final_moments
  none
  | standard
  | central
  ]
  [ distribution
  cumulative
  | complementary
  ]
  )
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( importance_sampling ALIAS nond_importance_sampling
[ samples ALIAS initial_samples INTEGER ]
[ seed INTEGER > 0 ]
import
| adapt_import
| mm_adapt_import
[ refinement_samples INTEGERLIST ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
  series
  | parallel
  ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary

```



```

    ]
  [ rng
    mt19937
    | rnum2
  ]
  [ model_pointer STRING ]
)
|
( gpais ALIAS gaussian_process_adaptive_importance_sampling
  [ build_samples ALIAS samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ samples_on_emulator INTEGER ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  [ active_only ]
  ]
  [ export_approx_points_file ALIAS export_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
  [ max_iterations INTEGER >= 0 ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | gen_reliabilities
      [ system
        series
        | parallel
      ]
    ]
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ]
  ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ rng
    mt19937
    | rnum2
  ]
  [ model_pointer STRING ]
)
|
( adaptive_sampling ALIAS nond_adaptive_sampling
  [ initial_samples ALIAS samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ samples_on_emulator INTEGER ]
  [ fitness_metric
    predicted_variance
    | distance
    | gradient
  ]
  [ batch_selection
    naive
    | distance_penalty
    | topology
  ]
)

```

```

    | constant_liar
  ]
[ refinement_samples INTEGERLIST ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ misc_options STRINGLIST ]
[ max_iterations INTEGER >= 0 ]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
    series
    | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( pof_darts ALIAS nond_pof_darts
  build_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ lipschitz
    local
    | global
  ]
  [ samples_on_emulator INTEGER ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | gen_reliabilities
      [ system
      series
      | parallel
      ]
    ]
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ]
  ]
)

```

```

]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( rkd_darts ALIAS nond_rkd_darts
  build_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ lipschitz
    local
    | global
  ]
  [ samples_on_emulator INTEGER ]
  [ response_levels REALLIST
    [ num_response_levels INTEGERLIST ]
    [ compute
      probabilities
      | gen_reliabilities
    ]
    [ system
      series
      | parallel
    ]
  ]
  [ probability_levels REALLIST
    [ num_probability_levels INTEGERLIST ]
  ]
  [ gen_reliability_levels REALLIST
    [ num_gen_reliability_levels INTEGERLIST ]
  ]
  [ distribution
    cumulative
    | complementary
  ]
  [ rng
    mt19937
    | rnum2
  ]
  [ model_pointer STRING ]
)
|
( global_evidence ALIAS nond_global_evidence
  [ samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ ( sbgo
    [ gaussian_process ALIAS kriging
      surfpack
      | dakota
    ]
    ( experimental
      [ options_file STRING ]
    )
  ]
  [ use_derivatives ]
  [ import_build_points_file ALIAS import_points_file STRING ]
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]

```

```

]
[ export_approx_points_file ALIAS export_points_file STRING
[ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
| annotated
| freeform ]
]
)
|
( ego
[ gaussian_process ALIAS kriging
( surfpack
[ export_model
[ filename_prefix STRING ]
( formats
[ text_archive ]
[ binary_archive ]
)
]
)
| dakota
|
( experimental
[ export_model
[ filename_prefix STRING ]
( formats
[ text_archive ]
[ binary_archive ]
)
]
[ options_file STRING ]
)
]
[ use_derivatives ]
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
| annotated
| freeform ]
[ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
[ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
| annotated
| freeform ]
]
]
| ea
| lhs ]
[ response_levels REALLIST
[ num_response_levels INTEGERLIST ]
[ compute
probabilities
| gen_reliabilities
[ system
series
| parallel
]
]
]
[ probability_levels REALLIST
[ num_probability_levels INTEGERLIST ]
]
]

```

```

[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( global_interval_est ALIAS nond_global_interval_est
[ samples INTEGER ]
[ seed INTEGER > 0 ]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ max_function_evaluations INTEGER >= 0 ]
[ ( sbgo
  [ gaussian_process ALIAS kriging
  surfpack
  | dakota
  |
  ( experimental
  [ options_file STRING ]
  )
]
  [ use_derivatives ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
  [ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
]
)
|
( ego
[ gaussian_process ALIAS kriging
  ( surfpack
  [ export_model
  [ filename_prefix STRING ]
  ( formats
  [ text_archive ]
  [ binary_archive ]
  )
]
]
)
| dakota
|
( experimental
[ export_model
[ filename_prefix STRING ]
( formats
[ text_archive ]
[ binary_archive ]
)
]
]
[ options_file STRING ]

```

```

)
]
[ use_derivatives ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
[ active_only ]
]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
]
| ea
| lhs ]
[ rng
  mt19937
  | rnum2
]
[ model_pointer STRING ]
)
|
( bayes_calibration ALIAS nond_bayes_calibration
  ( queso
    chain_samples ALIAS samples INTEGER
    [ seed INTEGER > 0 ]
    [ rng
      mt19937
      | rnum2
    ]
    [ emulator
      ( gaussian_process ALIAS kriging
        surfpack
        | dakota
        [ build_samples INTEGER ]
        [ posterior_adaptive ]
        [ import_build_points_file ALIAS import_points_file STRING
          [ ( custom_annotated
            [ header ]
            [ eval_id ]
            [ interface_id ]
          )
          | annotated
          | freeform ]
          [ active_only ]
        ]
      )
    ]
  )
  |
  ( pce
  [ p_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | decay
      | generalized
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  ( quadrature_order INTEGER
    [ dimension_preference REALLIST ]
    [ nested
      | non_nested ]
  )
)

```

```

)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
| cubature_integrand INTEGER
|
( expansion_order INTEGER
  [ dimension_preference REALLIST ]
  [ basis_type
    tensor_product
    | total_order
  ]
  ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
)
|
( collocation_points INTEGER
  [ ( least_squares
  | svd
  | equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
)
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
|
( least_angle_regression ALIAS lars
  [ noise_tolerance REALLIST ]
)
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
)
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( collocation_ratio REAL
  [ ( least_squares
  | svd
  | equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
)
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
)
|
( least_angle_regression ALIAS lars

```

```

    [ noise_tolerance REALLIST ]
  ]
|
( least_absolute_shrinkage ALIAS lasso
  [ noise_tolerance REALLIST ]
  [ l2_penalty REAL ]
  ]
[ cross_validation
  [ noise_only ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  ]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples INTEGER
  [ reuse_points ALIAS reuse_samples ]
  )
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
  [ active_only ]
  ]
[ posterior_adaptive ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points INTEGER
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
    | annotated
    | freeform ]
    [ active_only ]
    ]
  [ posterior_adaptive ]
  )
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( ml_pce
[ allocation_control
  ( estimator_variance
    [ estimator_rate REAL ]
    )
  | rip_sampling
  ]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
  ]
( expansion_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ basis_type

```



```

    tensor_product
    | total_order
    |
    ( adapted
      [ advancements INTEGER ]
      [ soft_convergence_limit INTEGER ]
    )
  ]
( collocation_ratio REAL
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ ( least_squares
    [ svd
    | equality_constrained ]
    )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  ]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  ]
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  ]
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  ]
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)

```

```

)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_pce
[ p_refinement
uniform
|
( dimension_adaptive
sobol
| decay
| generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
greedy
]
[ discrepancy_emulation
distinct ALIAS paired
| recursive
]
( quadrature_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ nested
| non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ restricted
| unrestricted ]
[ nested
| non_nested ]
)
|
( expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
tensor_product
| total_order
|
( adapted
[ advancements INTEGER ]
[ soft_convergence_limit INTEGER ]
)
]
( collocation_ratio REAL
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ ( least_squares
| svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso

```

```

    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  ]
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( sc
  [ ( p_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | generalized
    )
  )
)
)
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
)
[ max_refinement_iterations INTEGER >= 0 ]

```

```

( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
  greedy
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
]

```

```

[ standardized_space ]
[ logit_transform ]
[ export_chain_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ dram
| delayed_rejection
| adaptive_metropolis
| metropolis_hastings
| multilevel ]
[ pre_solve
  sqp
  | nip
  | none
]
[ proposal_covariance
  ( prior
[ multiplier REAL > 0.0 ]
)
  |
  ( derivatives
[ update_period INTEGER ]
)
  |
  ( values REALLIST
diagonal
| matrix
)
  |
  ( filename STRING
diagonal
| matrix
)
  ]
[ options_file STRING ]
)
|
( gpmsa
chain_samples ALIAS samples INTEGER
[ seed INTEGER > 0 ]
[ rng
  mt19937
  | rnum2
]
build_samples INTEGER
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ standardized_space ]
[ logit_transform ]
[ gpmsa_normalize ]
[ export_chain_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
  | annotated
  | freeform ]
]
[ dram

```

```

| delayed_rejection
| adaptive_metropolis
| metropolis_hastings ]
[ proposal_covariance
  ( prior
[ multiplier REAL > 0.0 ]
)
|
  ( derivatives
[ update_period INTEGER ]
)
|
  ( values REALLIST
diagonal
| matrix
)
|
  ( filename STRING
diagonal
| matrix
)
]
[ options_file STRING ]
)
|
( wasabi
pushforward_samples INTEGER
[ seed INTEGER > 0 ]
[ emulator
  ( gaussian_process ALIAS kriging
surfpack
| dakota
[ build_samples INTEGER ]
[ posterior_adaptive ]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
| annotated
| freeform ]
[ active_only ]
]
)
|
  ( pce
[ p_refinement
uniform
|
  ( dimension_adaptive
sobel
| decay
| generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
( quadrature_order INTEGER
[ dimension_preference REALLIST ]
[ nested
| non_nested ]
)
|
( sparse_grid_level INTEGER
[ dimension_preference REALLIST ]
[ restricted
| unrestricted ]
[ nested
| non_nested ]
)
| cubature_integrand INTEGER
|
( expansion_order INTEGER

```

```

[ dimension_preference REALLIST ]
[ basis_type
  tensor_product
  | total_order
  |
  ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
]
( collocation_points INTEGER
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  ]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  ]
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  ]
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  ]
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( collocation_ratio REAL
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  ]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  ]
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  ]
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  ]
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]

```

```

    [ response_scaling ]
    [ use_derivatives ]
    [ tensor_grid ]
    [ reuse_points ALIAS reuse_samples ]
    [ max_solver_iterations INTEGER >= 0 ]
  )
|
( expansion_samples INTEGER
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
[ posterior_adaptive ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points INTEGER
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
  [ posterior_adaptive ]
)
|
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( ml_pce
[ allocation_control
  ( estimator_variance
    [ estimator_rate REAL ]
  )
  | rip_sampling
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
( expansion_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ basis_type
    tensor_product
    | total_order
  ]
  |
  ( adapted
    [ advancements INTEGER ]
    [ soft_convergence_limit INTEGER ]
  )
]
)
( collocation_ratio REAL
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ ( least_squares
    [ svd

```



```

    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  )
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  )
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
  )
  |
  ( expansion_samples_sequence INTEGERLIST
    [ reuse_points ALIAS reuse_samples ]
  )
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
  )
  |
  ( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
    collocation_points_sequence INTEGERLIST
    [ tensor_grid INTEGERLIST ]
    [ reuse_points ALIAS reuse_samples ]
    [ import_build_points_file ALIAS import_points_file STRING
      [ ( custom_annotated
        [ header ]
        [ eval_id ]
        [ interface_id ]
      )
      | annotated
      | freeform ]
      [ active_only ]
    ]
  )
  [ askey
  | wiener ]
  [ normalized ]
  [ export_expansion_file STRING ]
  [ diagonal_covariance
  | full_covariance ]
  )
  |
  ( mf_pce
  [ p_refinement
  uniform

```

```

|
( dimension_adaptive
  sobol
  | decay
  | generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
  greedy
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
|
( expansion_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ basis_type
    tensor_product
    | total_order
    |
    ( adapted
      [ advancements INTEGER ]
      [ soft_convergence_limit INTEGER ]
    )
  ]
)
( collocation_ratio REAL
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ ( least_squares
    [ svd
    | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
    [ noise_tolerance REALLIST ]
  )
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_angle_regression ALIAS lars
    [ noise_tolerance REALLIST ]
  )
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  )
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]

```

```

    [ max_solver_iterations INTEGER >= 0 ]
  )
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
|
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
]
)
[ max_refinement_iterations INTEGER >= 0 ]
( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
)

```

```

    [ nested
    | non_nested ]
  )
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
)
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
| local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
  greedy
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
]
[ standardized_space ]
( data_distribution
  ( gaussian
  means REALLIST
  ( covariance REALLIST
    diagonal
    | matrix
  )
)
)
| obs_data_filename STRING
)
[ posterior_samples_import_filename STRING ]

```

```

[ generate_posterior_samples
  [ posterior_samples_export_filename STRING ]
]
[ evaluate_posterior_density
  [ posterior_density_export_filename STRING ]
]
)
|
( dream
  chain_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ chains INTEGER >= 3 ]
  [ num_cr INTEGER >= 1 ]
  [ crossover_chain_pairs INTEGER >= 0 ]
  [ gr_threshold REAL > 0.0 ]
  [ jump_step INTEGER >= 0 ]
  [ emulator
    ( gaussian_process ALIAS kriging
      surfpack
      | dakota
      [ build_samples INTEGER ]
      [ posterior_adaptive ]
      [ import_build_points_file ALIAS import_points_file STRING
        [ ( custom_annotated
          [ header ]
          [ eval_id ]
          [ interface_id ]
        )
        | annotated
        | freeform ]
      [ active_only ]
    ]
  )
  |
  ( pce
  [ p_refinement
    uniform
    |
    ( dimension_adaptive
      sobol
      | decay
      | generalized
    )
  ]
  [ max_refinement_iterations INTEGER >= 0 ]
  ( quadrature_order INTEGER
    [ dimension_preference REALLIST ]
    [ nested
      | non_nested ]
  )
  |
  ( sparse_grid_level INTEGER
    [ dimension_preference REALLIST ]
    [ restricted
      | unrestricted ]
    [ nested
      | non_nested ]
  )
  | cubature_integrand INTEGER
  |
  ( expansion_order INTEGER
    [ dimension_preference REALLIST ]
    [ basis_type
      tensor_product
      | total_order
    ]
    |
    ( adapted
      [ advancements INTEGER ]
      [ soft_convergence_limit INTEGER ]
    )
  ]
  ( collocation_points INTEGER
    [ ( least_squares

```

```

[ svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
| cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
| [ ratio_order REAL ]
| [ response_scaling ]
| [ use_derivatives ]
| [ tensor_grid ]
| [ reuse_points ALIAS reuse_samples ]
| [ max_solver_iterations INTEGER >= 0 ]
)
|
( collocation_ratio REAL
[ ( least_squares
[ svd
| equality_constrained ]
)
]
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
| cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
| [ ratio_order REAL ]
| [ response_scaling ]
| [ use_derivatives ]
| [ tensor_grid ]
| [ reuse_points ALIAS reuse_samples ]
| [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples INTEGER
[ reuse_points ALIAS reuse_samples ]
)
| [ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated

```

```

    [ header ]
    [ eval_id ]
    [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
    ]
  [ posterior_adaptive ]
  )
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points INTEGER
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
      )
      | annotated
      | freeform ]
      [ active_only ]
      ]
    [ posterior_adaptive ]
    )
  [ askey
  | wiener ]
  [ normalized ]
  [ export_expansion_file STRING ]
  [ diagonal_covariance
  | full_covariance ]
  )
|
  ( ml_pce
  [ allocation_control
    ( estimator_variance
      [ estimator_rate REAL ]
      )
    | rip_sampling
    ]
  [ discrepancy_emulation
  distinct ALIAS paired
  | recursive
  ]
  ( expansion_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ basis_type
    tensor_product
    | total_order
    |
    ( adapted
      [ advancements INTEGER ]
      [ soft_convergence_limit INTEGER ]
      )
    ]
  )
  [ collocation_ratio REAL
  [ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
  [ ( least_squares
  [ svd
  | equality_constrained ]
  )
  |
  ( orthogonal_matching_pursuit ALIAS omp
  [ noise_tolerance REALLIST ]
  ]
  | basis_pursuit ALIAS bp
  |
  ( basis_pursuit_denoising ALIAS bpdn
  [ noise_tolerance REALLIST ]
  ]
  ]
|

```

```

    ( least_angle_regression ALIAS lars
      [ noise_tolerance REALLIST ]
    )
  |
  ( least_absolute_shrinkage ALIAS lasso
    [ noise_tolerance REALLIST ]
    [ l2_penalty REAL ]
  )
  [ cross_validation
    [ noise_only ]
    [ max_cv_order_candidates INTEGER >= 0 ]
  ]
  [ ratio_order REAL ]
  [ response_scaling ]
  [ use_derivatives ]
  [ tensor_grid ]
  [ reuse_points ALIAS reuse_samples ]
  [ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
  [ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_pce
[ p_refinement
uniform
|
( dimension_adaptive
sobol
| decay
| generalized
)
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
greedy
]
[ discrepancy_emulation

```



```

distinct ALIAS paired
| recursive
]
( quadrature_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ nested
| non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ restricted
| unrestricted ]
[ nested
| non_nested ]
)
|
( expansion_order_sequence INTEGERLIST
[ dimension_preference REALLIST ]
[ basis_type
tensor_product
| total_order
|
( adapted
[ advancements INTEGER ]
[ soft_convergence_limit INTEGER ]
)
]
( collocation_ratio REAL
[ collocation_points_sequence ALIAS pilot_samples INTEGERLIST ]
[ ( least_squares
[ svd
| equality_constrained ]
)
|
( orthogonal_matching_pursuit ALIAS omp
[ noise_tolerance REALLIST ]
]
| basis_pursuit ALIAS bp
|
( basis_pursuit_denoising ALIAS bpdn
[ noise_tolerance REALLIST ]
]
|
( least_angle_regression ALIAS lars
[ noise_tolerance REALLIST ]
]
|
( least_absolute_shrinkage ALIAS lasso
[ noise_tolerance REALLIST ]
[ l2_penalty REAL ]
]
[ cross_validation
[ noise_only ]
[ max_cv_order_candidates INTEGER >= 0 ]
]
[ ratio_order REAL ]
[ response_scaling ]
[ use_derivatives ]
[ tensor_grid ]
[ reuse_points ALIAS reuse_samples ]
[ max_solver_iterations INTEGER >= 0 ]
)
|
( expansion_samples_sequence INTEGERLIST
[ reuse_points ALIAS reuse_samples ]
)
[ import_build_points_file ALIAS import_points_file STRING
[ ( custom_annotated
[ header ]
[ eval_id ]
[ interface_id ]
)
]
)

```

```

    | annotated
    | freeform ]
    [ active_only ]
  ]
)
|
( orthogonal_least_interpolation ALIAS least_interpolation ALIAS oli
  collocation_points_sequence INTEGERLIST
  [ tensor_grid INTEGERLIST ]
  [ reuse_points ALIAS reuse_samples ]
  [ import_build_points_file ALIAS import_points_file STRING
    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
    [ active_only ]
  ]
)
[ askey
| wiener ]
[ normalized ]
[ export_expansion_file STRING ]
[ diagonal_covariance
| full_covariance ]
)
|
( sc
[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
]
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
( quadrature_order INTEGER
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level INTEGER
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
|
( mf_sc

```

```

[ ( p_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
)
|
( h_refinement
  uniform
  |
  ( dimension_adaptive
    sobol
    | generalized
  )
  | local_adaptive
]
[ max_refinement_iterations INTEGER >= 0 ]
[ allocation_control
  greedy
]
[ discrepancy_emulation
  distinct ALIAS paired
  | recursive
]
( quadrature_order_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nested
  | non_nested ]
)
|
( sparse_grid_level_sequence INTEGERLIST
  [ dimension_preference REALLIST ]
  [ nodal
  | hierarchical ]
  [ restricted
  | unrestricted ]
  [ nested
  | non_nested ]
)
[ piecewise
| askey
| wiener ]
[ use_derivatives ]
[ diagonal_covariance
| full_covariance ]
)
]
[ standardized_space ]
[ export_chain_points_file STRING
  [ ( custom_annotated
  [ header ]
  [ eval_id ]
  [ interface_id ]
  )
  | annotated
  | freeform ]
]
)
|
( muq
  chain_samples ALIAS samples INTEGER
  [ seed INTEGER > 0 ]
  [ rng
    mt19937
    | rnum2
  ]
  [ dram
  | delayed_rejection
  | adaptive_metropolis
  | metropolis_hastings ]
  [ proposal_covariance

```

```

    ( prior
  [ multiplier REAL > 0.0 ]
  )
  |
  ( derivatives
  [ update_period INTEGER ]
  )
  |
  ( values REALLIST
diagonal
  | matrix
  )
  |
  ( filename STRING
diagonal
  | matrix
  )
  ]
  )
[ experimental_design
initial_samples ALIAS samples INTEGER
num_candidates INTEGER > 0
[ max_hifi_evaluations INTEGER >= 0 ]
[ batch_size INTEGER >= 1 ]
[ import_candidate_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  ]
[ ksg2 ]
]
[ calibrate_error_multipliers
one
  | per_experiment
  | per_response
  | both
  [ hyperprior_alphas REALLIST
  hyperprior_betas REALLIST
  ]
]
[ burn_in_samples INTEGER ]
[ posterior_stats
  [ kl_divergence ]
  [ mutual_info
    [ ksg2 ]
  ]
  [ kde ]
]
[ chain_diagnostics
  [ confidence_intervals ]
]
[ model_evidence
  [ mc_approx ]
  [ evidence_samples INTEGER ]
  [ laplace_approx ]
]
[ model_discrepancy
  [ discrepancy_type
    ( gaussian_process ALIAS kriging
  [ trend_order INTEGER >= 0 ]
  )
  |
  ( polynomial
  [ basis_order INTEGER >= 0 ]
  )
  ]
  [ num_prediction_configs INTEGER >= 0 ]
  [ prediction_configs REALLIST ]
  [ import_prediction_configs STRING

```

```

    [ ( custom_annotated
      [ header ]
      [ eval_id ]
      [ interface_id ]
    )
    | annotated
    | freeform ]
  ]
[ export_discrepancy_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ export_corrected_model_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ export_corrected_variance_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
]
[ sub_sampling_period INTEGER ]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ convergence_tolerance REAL ]
[ max_iterations INTEGER >= 0 ]
[ model_pointer STRING ]
[ scaling ]
)
|
( dace
  grid
  | random
  | oas
  | lhs
  | oa_lhs
  | box_behnken
  | central_composite
  [ samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ fixed_seed ]
  [ main_effects ]
  [ quality_metrics ]
  [ variance_based_decomp
    [ drop_tolerance REAL ]
  ]
  [ symbols INTEGER ]
  [ model_pointer STRING ]
)
|
( fsu_cvt
  [ samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ fixed_seed ]
  [ latinize ]
  [ quality_metrics ]
  [ variance_based_decomp

```

```

    [ drop_tolerance REAL ]
  ]
[ trial_type
  grid
  | halton
  | random
]
[ num_trials INTEGER ]
[ max_iterations INTEGER >= 0 ]
[ model_pointer STRING ]
)
|
( psuade_moat
  [ partitions INTEGERLIST ]
  [ samples INTEGER ]
  [ seed INTEGER > 0 ]
  [ model_pointer STRING ]
)
|
( local_evidence ALIAS nond_local_evidence
  [ sqp
  | nip ]
  [ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
    series
    | parallel
    ]
  ]
  ]
  [ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
  ]
  [ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
  ]
  [ distribution
  cumulative
  | complementary
  ]
  [ model_pointer STRING ]
)
|
( local_interval_est ALIAS nond_local_interval_est
  [ sqp
  | nip ]
  [ convergence_tolerance REAL ]
  [ model_pointer STRING ]
)
|
( local_reliability ALIAS nond_local_reliability
  [ mpp_search
  x_taylor_mean
  | u_taylor_mean
  | x_taylor_mpp
  | u_taylor_mpp
  | x_two_point
  | u_two_point
  | x_multi_point
  | u_multi_point
  | no_approx
  [ sqp
  | nip ]
  [ integration
  first_order
  | second_order
  [ probability_refinement ALIAS sample_refinement
  import
  | adapt_import
  | mm_adapt_import

```

```

    [ refinement_samples INTEGERLIST ]
    [ seed INTEGER > 0 ]
  ]
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | reliabilities
    | gen_reliabilities
    [ system
      series
      | parallel
    ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ reliability_levels REALLIST
  [ num_reliability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ final_moments
  none
  | standard
  | central
]
[ model_pointer STRING ]
)
|
( global_reliability ALIAS nond_global_reliability
  [ initial_samples INTEGER ]
  x_gaussian_process ALIAS x_kriging
  | u_gaussian_process ALIAS u_kriging
  [ ( surfpack
    [ export_model
    [ filename_prefix STRING ]
    ( formats
      [ text_archive ]
      [ binary_archive ]
    )
  )
]
)
| dakota
|
( experimental
  [ export_model
    [ filename_prefix STRING ]
    ( formats
      [ text_archive ]
      [ binary_archive ]
    )
  ]
  [ options_file STRING ]
]
[ import_build_points_file ALIAS import_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]

```

```

    [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
]
[ use_derivatives ]
[ seed INTEGER > 0 ]
[ rng
  mt19937
  | rnum2
]
[ response_levels REALLIST
  [ num_response_levels INTEGERLIST ]
  [ compute
    probabilities
    | gen_reliabilities
    [ system
  series
  | parallel
  ]
  ]
]
[ probability_levels REALLIST
  [ num_probability_levels INTEGERLIST ]
]
[ gen_reliability_levels REALLIST
  [ num_gen_reliability_levels INTEGERLIST ]
]
[ distribution
  cumulative
  | complementary
]
[ max_iterations INTEGER >= 0 ]
[ convergence_tolerance REAL ]
[ model_pointer STRING ]
)
|
( fsu_quasi_mc
  halton
  | hammersley
  [ latinize ]
  [ quality_metrics ]
  [ variance_based_decomp
    [ drop_tolerance REAL ]
  ]
  [ samples INTEGER ]
  [ fixed_sequence ]
  [ sequence_start INTEGERLIST ]
  [ sequence_leap INTEGERLIST ]
  [ prime_base INTEGERLIST ]
  [ max_iterations INTEGER >= 0 ]
  [ model_pointer STRING ]
)
|
( vector_parameter_study
  final_point REALLIST
  | step_vector REALLIST
  num_steps INTEGER
  [ model_pointer STRING ]
)
|
( list_parameter_study
  list_of_points REALLIST
  |
  ( import_points_file STRING
    [ ( custom_annotated
      [ header ]

```



```

    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  [ active_only ]
  )
  [ model_pointer STRING ]
  )
|
( centered_parameter_study
  step_vector REALLIST
  steps_per_variable ALIAS deltas_per_variable INTEGERLIST
  [ model_pointer STRING ]
  )
|
( multidim_parameter_study
  partitions INTEGERLIST
  [ model_pointer STRING ]
  )
|
( richardson_extrap
  estimate_order
  | converge_order
  | converge_qoi
  [ refinement_rate REAL ]
  [ convergence_tolerance REAL ]
  [ max_iterations INTEGER >= 0 ]
  [ model_pointer STRING ]
  )

```

KEYWORD model

```

  [ id_model STRING ]
  ( single ALIAS simulation
    [ interface_pointer STRING ]
    [ solution_level_control STRING ]
    [ solution_level_cost REALLIST
    | cost_recovery_metadata STRING ]
    )
  |
  ( surrogate
    [ id_surrogates INTEGERLIST ]
    ( global
      ( experimental_gaussian_process
        [ trend
        none
        | constant
        | linear
        | reduced_quadratic
        | quadratic
        ]
        [ num_restarts INTEGER > 1 ]
        [ nugget REAL > 0
        | find_nugget INTEGER ]
        [ options_file STRING ]
        [ export_approx_variance_file STRING
        [ ( custom_annotated
          [ header ]
          [ eval_id ]
          [ interface_id ]
          )
        | annotated
        | freeform ]
        ]
        [ export_model
        [ filename_prefix STRING ]
        ( formats
          [ text_archive ]
          [ binary_archive ]
          )
        ]
        [ import_model
        [ filename_prefix STRING ]

```

```

text_archive
| binary_archive
]
)
|
( gaussian_process ALIAS kriging
  ( dakota
  [ point_selection ]
  [ trend
    constant
    | linear
    | reduced_quadratic
  ]
  )
  |
  ( surfpack
  [ trend
    constant
    | linear
    | reduced_quadratic
    | quadratic
  ]
  [ optimization_method STRING ]
  [ max_trials INTEGER > 0 ]
  [ nugget REAL > 0
  | find_nugget INTEGER ]
  [ correlation_lengths REALLIST ]
  [ export_model
  [ filename_prefix STRING ]
  ( formats
    [ text_archive ]
    [ binary_archive ]
    [ algebraic_file ]
    [ algebraic_console ]
  )
  ]
  [ import_model
  [ filename_prefix STRING ]
  text_archive
  | binary_archive
  ]
  )
  [ export_approx_variance_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  ]
  )
  |
  ( mars
  [ max_bases INTEGER ]
  [ interpolation
linear
| cubic
]
  [ export_model
  [ filename_prefix STRING ]
  ( formats
    [ text_archive ]
    [ binary_archive ]
  )
  ]
  [ import_model
  [ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|

```

```

( moving_least_squares
  [ basis_order ALIAS poly_order INTEGER >= 0 ]
  [ weight_function INTEGER ]
  [ export_model
  [ filename_prefix STRING ]
  ( formats
    [ text_archive ]
    [ binary_archive ]
  )
]
  [ import_model
  [ filename_prefix STRING ]
text_archive
| binary_archive
]
  )
|
( function_train
  [ regression_type
ls
|
  ( rls2
    l2_penalty REAL
  )
]
  [ max_solver_iterations INTEGER >= 0 ]
  [ max_cross_iterations INTEGER >= 0 ]
  [ solver_tolerance REAL ]
  [ response_scaling ]
  [ tensor_grid ]
  [ rounding_tolerance REAL ]
  [ arithmetic_tolerance REAL ]
  [ start_order ALIAS order INTEGER >= 0 ]
  [ dimension_preference REALLIST ]
]
  [ adapt_order ]
  [ kick_order INTEGER > 0 ]
  [ max_order INTEGER >= 0 ]
  [ max_cv_order_candidates INTEGER >= 0 ]
  [ start_rank ALIAS rank INTEGER >= 0 ]
  [ adapt_rank ]
  [ kick_rank INTEGER > 0 ]
  [ max_rank INTEGER >= 0 ]
  [ max_cv_rank_candidates INTEGER >= 0 ]
  )
|
( neural_network
  [ max_nodes ALIAS nodes INTEGER ]
  [ range REAL ]
  [ random_weight INTEGER ]
  [ export_model
  [ filename_prefix STRING ]
  ( formats
    [ text_archive ]
    [ binary_archive ]
    [ algebraic_file ]
    [ algebraic_console ]
  )
]
  [ import_model
  [ filename_prefix STRING ]
text_archive
| binary_archive
]
  )
|
( radial_basis
  [ bases INTEGER ]
  [ max_pts INTEGER ]
  [ min_partition INTEGER ]
  [ max_subsets INTEGER ]
  [ export_model
  [ filename_prefix STRING ]

```

```

( formats
  [ text_archive ]
  [ binary_archive ]
  [ algebraic_file ]
  [ algebraic_console ]
)
]
[ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( polynomial
basis_order INTEGER >= 0
| linear
| quadratic
| cubic
[ export_model
[ filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
  [ algebraic_file ]
  [ algebraic_console ]
)
]
[ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
|
( experimental_polynomial
basis_order INTEGER >= 0
[ options_file STRING ]
[ export_model
[ filename_prefix STRING ]
( formats
  [ text_archive ]
  [ binary_archive ]
)
]
[ import_model
[ filename_prefix STRING ]
text_archive
| binary_archive
]
)
[ domain_decomposition
  [ cell_type STRING ]
  [ support_layers INTEGER ]
  [ discontinuity_detection
jump_threshold REAL
| gradient_threshold REAL
]
]
[ total_points INTEGER
| minimum_points
| recommended_points ]
[ ( dace_method_pointer STRING
[ auto_refinement
  [ max_iterations INTEGER > 0 ]
  [ max_function_evaluations INTEGER > 0 ]
  [ convergence_tolerance REAL ]
  [ soft_convergence_limit INTEGER >= 0 ]
  [ cross_validation_metric STRING
  [ folds INTEGER > 0 ]
  ]
]
]
)

```

```

| truth_model_pointer ALIAS actual_model_pointer STRING ]
[ reuse_points ALIAS reuse_samples
  all
  | region
  | none
  ]
[ import_build_points_file ALIAS import_points_file ALIAS samples_file STRING
  [ ( custom_annotated
    [ header
      [ use_variable_labels ]
    ]
    [ eval_id ]
    [ interface_id ]
  )
  |
  ( annotated
  [ use_variable_labels ]
  ]
  | freeform ]
  [ active_only ]
  ]
[ export_approx_points_file ALIAS export_points_file STRING
  [ ( custom_annotated
    [ header ]
    [ eval_id ]
    [ interface_id ]
  )
  | annotated
  | freeform ]
  ]
[ use_derivatives ]
[ correction
  zeroth_order
  | first_order
  | second_order
  additive
  | multiplicative
  | combined
  ]
[ metrics ALIAS diagnostics STRINGLIST
  [ cross_validation
  [ folds INTEGER
  | percent REAL ]
  ]
  [ press ]
  ]
[ import_challenge_points_file ALIAS challenge_points_file STRING
  [ ( custom_annotated
    [ header
      [ use_variable_labels ]
    ]
    [ eval_id ]
    [ interface_id ]
  )
  |
  ( annotated
  [ use_variable_labels ]
  ]
  | freeform ]
  [ active_only ]
  ]
)
|
( multipoint
  tana
  | qmea
  truth_model_pointer ALIAS actual_model_pointer STRING
  )
|
( local
  taylor_series
  truth_model_pointer ALIAS actual_model_pointer STRING
  )

```

```

|
( hierarchical
  ordered_model_fidelities ALIAS model_fidelity_sequence STRINGLIST
  [ correction
    zeroth_order
    | first_order
    | second_order
    additive
    | multiplicative
    | combined
  ]
)
|
( non_hierarchical ALIAS model_ensemble
  truth_model_pointer ALIAS actual_model_pointer STRING
  [ unordered_model_fidelities ALIAS approximation_models STRINGLIST ]
)
)
|
( nested
  [ optional_interface_pointer STRING
    [ optional_interface_responses_pointer STRING ]
  ]
  ( sub_method_pointer STRING
    [ iterator_servers INTEGER > 0 ]
    [ iterator_scheduling
      master
      | peer
    ]
    [ processors_per_iterator INTEGER > 0 ]
    [ primary_variable_mapping STRINGLIST ]
    [ secondary_variable_mapping STRINGLIST ]
    [ primary_response_mapping REALLIST ]
    [ secondary_response_mapping REALLIST ]
    [ identity_response_mapping ]
  )
)
|
( active_subspace ALIAS subspace
  truth_model_pointer ALIAS actual_model_pointer STRING
  [ initial_samples INTEGER ]
  [ sample_type
    lhs
    | random
  ]
  [ truncation_method
    [ bing_li ]
    [ constantine ]
    [ energy
      [ truncation_tolerance REAL ]
    ]
    [ cross_validation
      [ minimum
        | relative
        | decrease ]
      [ relative_tolerance REAL ]
      [ decrease_tolerance REAL ]
      [ max_rank INTEGER ]
      [ exhaustive ]
    ]
  ]
  [ dimension INTEGER ]
  [ bootstrap_samples INTEGER ]
  [ build_surrogate
    [ refinement_samples INTEGERLIST ]
  ]
  [ normalization
    mean_value
    | mean_gradient
    | local_gradient
  ]
)
|

```

```

( adapted_basis
  ( truth_model_pointer ALIAS actual_model_pointer STRING
    sparse_grid_level INTEGER
    |
    ( expansion_order INTEGER
      collocation_ratio REAL
    )
    [ dimension INTEGER ]
    [ rotation_method
      unranked
      | ranked
    ]
  )
  [ truncation_tolerance REAL ]
)
|
( random_field
  [ build_source
    rf_data_file STRING
    | dace_method_pointer STRING
    |
    ( analytic_covariance
      squared_exponential
      | exponential
    )
  ]
  [ expansion_form
    karhunen_loeve
    | principal_components
  ]
  [ expansion_bases INTEGER ]
  [ truncation_tolerance REAL ]
  propagation_model_pointer STRING
)
[ variables_pointer STRING ]
[ responses_pointer STRING ]
[ hierarchical_tagging ]

KEYWORD variables
[ id_variables STRING ]
[ active
  all
  | design
  | uncertain
  | aleatory
  | epistemic
  | state
]
[ mixed
| relaxed ]
[ continuous_design INTEGER > 0
  [ initial_point ALIAS cdv_initial_point REALLIST ]
  [ lower_bounds ALIAS cdv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS cdv_upper_bounds REALLIST ]
  [ scale_types ALIAS cdv_scale_types STRINGLIST ]
  [ scales ALIAS cdv_scales REALLIST ]
  [ descriptors ALIAS cdv_descriptors STRINGLIST ]
]
[ discrete_design_range INTEGER > 0
  [ initial_point ALIAS ddv_initial_point INTEGERLIST ]
  [ lower_bounds ALIAS ddv_lower_bounds INTEGERLIST ]
  [ upper_bounds ALIAS ddv_upper_bounds INTEGERLIST ]
  [ descriptors ALIAS ddv_descriptors STRINGLIST ]
]
[ discrete_design_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ categorical STRINGLIST
      [ adjacency_matrix INTEGERLIST ]
    ]
  ]
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]

```

```

]
[ string INTEGER > 0
  [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
  elements ALIAS set_values STRINGLIST
  [ adjacency_matrix INTEGERLIST ]
  [ initial_point STRINGLIST ]
  [ descriptors STRINGLIST ]
]
[ real INTEGER > 0
  [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
  elements ALIAS set_values REALLIST
  [ categorical STRINGLIST
    [ adjacency_matrix INTEGERLIST ]
  ]
  [ initial_point REALLIST ]
  [ descriptors STRINGLIST ]
]
]
[ normal_uncertain INTEGER > 0
  means ALIAS nuv_means REALLIST
  std_deviations ALIAS nuv_std_deviations REALLIST
  [ lower_bounds ALIAS nuv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS nuv_upper_bounds REALLIST ]
  [ initial_point REALLIST ]
  [ descriptors ALIAS nuv_descriptors STRINGLIST ]
]
[ lognormal_uncertain INTEGER > 0
  ( lambdas ALIAS lnuv_lambdas REALLIST
    zetas ALIAS lnuv_zetas REALLIST
  )
  |
  ( means ALIAS lnuv_means REALLIST
    std_deviations ALIAS lnuv_std_deviations REALLIST
    | error_factors ALIAS lnuv_error_factors REALLIST
  )
  [ lower_bounds ALIAS lnuv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS lnuv_upper_bounds REALLIST ]
  [ initial_point REALLIST ]
  [ descriptors ALIAS lnuv_descriptors STRINGLIST ]
]
[ uniform_uncertain INTEGER > 0
  lower_bounds ALIAS uuv_lower_bounds REALLIST
  upper_bounds ALIAS uuv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS uuv_descriptors STRINGLIST ]
]
[ loguniform_uncertain INTEGER > 0
  lower_bounds ALIAS luuv_lower_bounds REALLIST
  upper_bounds ALIAS luuv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS luuv_descriptors STRINGLIST ]
]
[ triangular_uncertain INTEGER > 0
  modes ALIAS tuv_modes REALLIST
  lower_bounds ALIAS tuv_lower_bounds REALLIST
  upper_bounds ALIAS tuv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS tuv_descriptors STRINGLIST ]
]
[ exponential_uncertain INTEGER > 0
  betas ALIAS euv_betas REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS euv_descriptors STRINGLIST ]
]
[ beta_uncertain INTEGER > 0
  alphas ALIAS buv_alphas REALLIST
  betas ALIAS buv_betas REALLIST
  lower_bounds ALIAS buv_lower_bounds REALLIST
  upper_bounds ALIAS buv_upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS buv_descriptors STRINGLIST ]
]
]
[ gamma_uncertain INTEGER > 0

```



```

    alphas ALIAS gauv_alphas REALLIST
    betas ALIAS gauv_betas REALLIST
    [ initial_point REALLIST ]
    [ descriptors ALIAS gauv_descriptors STRINGLIST ]
  ]
[ gumbel_uncertain INTEGER > 0
  alphas ALIAS guuv_alphas REALLIST
  betas ALIAS guuv_betas REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS guuv_descriptors STRINGLIST ]
]
[ frechet_uncertain INTEGER > 0
  alphas ALIAS fuv_alphas REALLIST
  betas ALIAS fuv_betas REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS fuv_descriptors STRINGLIST ]
]
[ weibull_uncertain INTEGER > 0
  alphas ALIAS wuv_alphas REALLIST
  betas ALIAS wuv_betas REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS wuv_descriptors STRINGLIST ]
]
[ histogram_bin_uncertain INTEGER > 0
  [ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
  abscissas ALIAS huv_bin_abscissas REALLIST
  ordinates ALIAS huv_bin_ordinates REALLIST
  | counts ALIAS huv_bin_counts REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS huv_bin_descriptors STRINGLIST ]
]
[ poisson_uncertain INTEGER > 0
  lambdas REALLIST
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]
[ binomial_uncertain INTEGER > 0
  probability_per_trial ALIAS prob_per_trial REALLIST
  num_trials INTEGERLIST
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]
[ negative_binomial_uncertain INTEGER > 0
  probability_per_trial ALIAS prob_per_trial REALLIST
  num_trials INTEGERLIST
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]
[ geometric_uncertain INTEGER > 0
  probability_per_trial ALIAS prob_per_trial REALLIST
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]
[ hypergeometric_uncertain INTEGER > 0
  total_population INTEGERLIST
  selected_population INTEGERLIST
  num_drawn INTEGERLIST
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]
[ histogram_point_uncertain
  [ integer INTEGER > 0
    [ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
    abscissas INTEGERLIST
    counts REALLIST
    [ initial_point INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ string INTEGER > 0
    [ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
    abscissas STRINGLIST
    counts REALLIST
    [ initial_point STRINGLIST ]
  ]
]

```

```

    [ descriptors STRINGLIST ]
  ]
[ real INTEGER > 0
  [ pairs_per_variable ALIAS num_pairs INTEGERLIST ]
  abscissas REALLIST
  counts REALLIST
  [ initial_point REALLIST ]
  [ descriptors STRINGLIST ]
]
]
[ uncertain_correlation_matrix REALLIST ]
[ continuous_interval_uncertain ALIAS interval_uncertain INTEGER > 0
  [ num_intervals ALIAS iuv_num_intervals INTEGERLIST ]
  [ interval_probabilities ALIAS interval_probs ALIAS iuv_interval_probs REALLIST ]
  lower_bounds REALLIST
  upper_bounds REALLIST
  [ initial_point REALLIST ]
  [ descriptors ALIAS iuv_descriptors STRINGLIST ]
]
[ discrete_interval_uncertain INTEGER > 0
  [ num_intervals INTEGERLIST ]
  [ interval_probabilities ALIAS interval_probs ALIAS range_probabilities ALIAS range_probs REALLIST ]
  lower_bounds INTEGERLIST
  upper_bounds INTEGERLIST
  [ initial_point INTEGERLIST ]
  [ descriptors STRINGLIST ]
]
[ discrete_uncertain_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ set_probabilities ALIAS set_probs REALLIST ]
    [ categorical STRINGLIST ]
    [ initial_point INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ string INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values STRINGLIST
    [ set_probabilities ALIAS set_probs REALLIST ]
    [ initial_point STRINGLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ real INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values REALLIST
    [ set_probabilities ALIAS set_probs REALLIST ]
    [ categorical STRINGLIST ]
    [ initial_point REALLIST ]
    [ descriptors STRINGLIST ]
  ]
]
[ continuous_state INTEGER > 0
  [ initial_state ALIAS csv_initial_state REALLIST ]
  [ lower_bounds ALIAS csv_lower_bounds REALLIST ]
  [ upper_bounds ALIAS csv_upper_bounds REALLIST ]
  [ descriptors ALIAS csv_descriptors STRINGLIST ]
]
[ discrete_state_range INTEGER > 0
  [ initial_state ALIAS dsv_initial_state INTEGERLIST ]
  [ lower_bounds ALIAS dsv_lower_bounds INTEGERLIST ]
  [ upper_bounds ALIAS dsv_upper_bounds INTEGERLIST ]
  [ descriptors ALIAS dsv_descriptors STRINGLIST ]
]
[ discrete_state_set
  [ integer INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values INTEGERLIST
    [ categorical STRINGLIST ]
    [ initial_state INTEGERLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ string INTEGER > 0

```

```

    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values STRINGLIST
    [ initial_state STRINGLIST ]
    [ descriptors STRINGLIST ]
  ]
  [ real INTEGER > 0
    [ elements_per_variable ALIAS num_set_values INTEGERLIST ]
    elements ALIAS set_values REALLIST
    [ categorical STRINGLIST ]
    [ initial_state REALLIST ]
    [ descriptors STRINGLIST ]
  ]
]
[ linear_inequality_constraint_matrix REALLIST ]
[ linear_inequality_lower_bounds REALLIST ]
[ linear_inequality_upper_bounds REALLIST ]
[ linear_inequality_scale_types STRINGLIST ]
[ linear_inequality_scales REALLIST ]
[ linear_equality_constraint_matrix REALLIST ]
[ linear_equality_targets REALLIST ]
[ linear_equality_scale_types STRINGLIST ]
[ linear_equality_scales REALLIST ]

KEYWORD interface
  [ id_interface STRING ]
  [ analysis_drivers STRINGLIST
  [ input_filter STRING ]
  [ output_filter STRING ]
  ( system
    [ parameters_file STRING ]
    [ results_file STRING ]
    [ file_tag ]
    [ file_save ]
    [ labeled ]
    [ aprepro ALIAS dprepro ]
    [ work_directory
      [ named STRING ]
      [ directory_tag ALIAS dir_tag ]
      [ directory_save ALIAS dir_save ]
      [ link_files STRINGLIST ]
      [ copy_files STRINGLIST ]
      [ replace ]
    ]
    [ allow_existing_results ]
    [ verbatim ]
  )
  |
  ( fork
    [ parameters_file STRING ]
    [ results_file STRING ]
    [ file_tag ]
    [ file_save ]
    [ labeled ]
    [ aprepro ALIAS dprepro ]
    [ work_directory
      [ named STRING ]
      [ directory_tag ALIAS dir_tag ]
      [ directory_save ALIAS dir_save ]
      [ link_files STRINGLIST ]
      [ copy_files STRINGLIST ]
      [ replace ]
    ]
    [ allow_existing_results ]
    [ verbatim ]
  )
  |
  ( direct
    [ processors_per_analysis INTEGER > 0 ]
  )
  |
  ( plugin
    library_path STRING
  )

```

```

| matlab
|
( python
  [ numpy ]
)
|
( legacy_python
  [ numpy ]
)
| scilab
| grid
[ analysis_components STRINGLIST ]
]
[ algebraic_mappings STRING ]
[ failure_capture
  abort
  | retry INTEGER
  | recover REALLIST
  | continuation
]
[ deactivate
  [ active_set_vector ]
  [ evaluation_cache ]
  [ strict_cache_equality
    [ cache_tolerance REAL ]
  ]
  [ restart_file ]
]
[ ( batch
  [ size INTEGER > 0 ]
)
|
( asynchronous
  [ evaluation_concurrency INTEGER > 0 ]
  [ local_evaluation_scheduling
    dynamic
    | static
  ]
  [ analysis_concurrency INTEGER > 0 ]
]
[ evaluation_servers INTEGER > 0 ]
[ evaluation_scheduling
  master
  |
  ( peer
    dynamic
    | static
  )
]
[ processors_per_evaluation INTEGER > 0 ]
[ analysis_servers INTEGER > 0 ]
[ analysis_scheduling
  master
  | peer
]

```

KEYWORD responses

```

[ id_responses STRING ]
[ descriptors ALIAS response_descriptors STRINGLIST ]
( objective_functions ALIAS num_objective_functions INTEGER >= 0
  [ sense STRINGLIST ]
  [ primary_scale_types ALIAS objective_function_scale_types STRINGLIST ]
  [ primary_scales ALIAS objective_function_scales REALLIST ]
  [ weights ALIAS multi_objective_weights REALLIST ]
  [ nonlinear_inequality_constraints ALIAS num_nonlinear_inequality_constraints INTEGER >= 0
    [ lower_bounds ALIAS nonlinear_inequality_lower_bounds REALLIST ]
    [ upper_bounds ALIAS nonlinear_inequality_upper_bounds REALLIST ]
    [ scale_types ALIAS nonlinear_inequality_scale_types STRINGLIST ]
    [ scales ALIAS nonlinear_inequality_scales REALLIST ]
  ]
  [ nonlinear_equality_constraints ALIAS num_nonlinear_equality_constraints INTEGER >= 0
    [ targets ALIAS nonlinear_equality_targets REALLIST ]
    [ scale_types ALIAS nonlinear_equality_scale_types STRINGLIST ]
  ]
)

```

```

    [ scales ALIAS nonlinear_equality_scales REALLIST ]
  ]
  [ scalar_objectives ALIAS num_scalar_objectives INTEGER >= 0 ]
  [ field_objectives ALIAS num_field_objectives INTEGER >= 0
    lengths INTEGERLIST
    [ num_coordinates_per_field INTEGERLIST ]
    [ read_field_coordinates ]
  ]
)
|
( calibration_terms ALIAS least_squares_terms ALIAS num_least_squares_terms INTEGER >= 0
  [ scalar_calibration_terms INTEGER >= 0 ]
  [ field_calibration_terms INTEGER >= 0
    lengths INTEGERLIST
    [ num_coordinates_per_field INTEGERLIST ]
    [ read_field_coordinates ]
  ]
  [ primary_scales ALIAS calibration_term_scales ALIAS least_squares_term_scales REALLIST ]
  [ weights ALIAS calibration_weights ALIAS least_squares_weights REALLIST ]
  [ ( calibration_data
    [ data_directory STRING ]
    [ num_experiments INTEGER >= 0 ]
    [ num_config_variables INTEGER >= 0 ]
    [ experiment_variance_type ALIAS variance_type STRINGLIST ]
    [ scalar_data_file STRING
      [ ( custom_annotated
        [ header ]
        [ exp_id ]
      )
      | annotated
      | freeform ]
    ]
    [ interpolate ]
  )
  |
  ( calibration_data_file ALIAS least_squares_data_file STRING
    [ ( custom_annotated
      [ header ]
      [ exp_id ]
    )
    | annotated
    | freeform ]
    [ num_experiments INTEGER >= 0 ]
    [ num_config_variables INTEGER >= 0 ]
    [ experiment_variance_type ALIAS variance_type STRINGLIST ]
  ]
  [ simulation_variance REALLIST ]
  [ nonlinear_inequality_constraints ALIAS num_nonlinear_inequality_constraints INTEGER >= 0
    [ lower_bounds ALIAS nonlinear_inequality_lower_bounds REALLIST ]
    [ upper_bounds ALIAS nonlinear_inequality_upper_bounds REALLIST ]
    [ scale_types ALIAS nonlinear_inequality_scale_types STRINGLIST ]
    [ scales ALIAS nonlinear_inequality_scales REALLIST ]
  ]
  [ nonlinear_equality_constraints ALIAS num_nonlinear_equality_constraints INTEGER >= 0
    [ targets ALIAS nonlinear_equality_targets REALLIST ]
    [ scale_types ALIAS nonlinear_equality_scale_types STRINGLIST ]
    [ scales ALIAS nonlinear_equality_scales REALLIST ]
  ]
  )
  |
  ( response_functions ALIAS num_response_functions INTEGER >= 0
    [ scalar_responses ALIAS num_scalar_responses INTEGER >= 0 ]
    [ field_responses ALIAS num_field_responses INTEGER >= 0
      lengths INTEGERLIST
      [ num_coordinates_per_field INTEGERLIST ]
      [ read_field_coordinates ]
    ]
  )
  no_gradients
  | analytic_gradients
  |
  ( mixed_gradients
    id_numerical_gradients INTEGERLIST
  )

```

```

id_analytic_gradients INTEGERLIST
[ method_source ]
[ ( dakota
  [ ignore_bounds ]
  [ relative
  | absolute
  | bounds ]
  )
| vendor ]
[ interval_type ]
[ forward
| central ]
[ fd_step_size ALIAS fd_gradient_step_size REALLIST ]
)
|
( numerical_gradients
[ method_source ]
[ ( dakota
  [ ignore_bounds ]
  [ relative
  | absolute
  | bounds ]
  )
| vendor ]
[ interval_type ]
[ forward
| central ]
[ fd_step_size ALIAS fd_gradient_step_size REALLIST ]
)
no_hessians
|
( numerical_hessians
[ fd_step_size ALIAS fd_hessian_step_size REALLIST ]
[ relative
| absolute
| bounds ]
[ forward
| central ]
)
|
( quasi_hessians
( bfgs
[ damped ]
)
| srl
)
| analytic_hessians
|
( mixed_hessians
[ id_numerical_hessians INTEGERLIST
  [ fd_step_size ALIAS fd_hessian_step_size REALLIST ]
  ]
[ relative
| absolute
| bounds ]
[ forward
| central ]
[ id_quasi_hessians INTEGERLIST
  ( bfgs
  [ damped ]
  )
  | srl
  ]
[ id_analytic_hessians INTEGERLIST ]
)
[ metadata STRINGLIST ]

```

7.3.5 active_subspace

- [Keywords Area](#)

- [model](#)
- [active_subspace](#)

Active (variable) subspace model

Specification

Alias: subspace

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		truth_model_pointer	Pointer to specify a full-space model, from which to construct a lower dimensional surrogate
	Optional		initial_samples	Initial number of samples for sampling-based methods
	Optional		sample_type	Selection of sampling strategy
	Optional		truncation_method	Metric that estimates active subspace size
	Optional		dimension	Explicitly specify the desired subspace size
	Optional		bootstrap_samples	Number of bootstrap replicates used in truncation metrics
	Optional		build_surrogate	Construct moving least squares surrogate over active subspace

	Optional	normalization	Normalize gradient samples
--	-----------------	-------------------------------	----------------------------

Description

A model that transforms the original model (given by [truth_model_pointer](#)) to one with a reduced set of variables. This reduced model is identified by iteratively sampling the gradient of the original model and performing a singular value decomposition of the gradient matrix.

Expected Output

A subspace model will perform an initial sampling design to identify an active subspace using one of the truncation methods.

Usage Tips

If the desired subspace size is not identified, consider using the explicit [dimension](#) truncation option or one of the other truncation methods.

Examples

Perform an initial 100 gradient samples and use the [bing_li](#) truncation method to identify an active subspace. The truncation method uses 150 bootstrap samples to compute the Bing Li truncation metric.

```
model
  subspace
    id_model = 'SUBSPACE'
    truth_model_pointer = 'FULLSPACE'
    initial_samples 100
    truncation_method bing_li
    bootstrap_samples 150
```

Theory

The idea behind active subspaces is to find directions in the input variable space in which the quantity of interest is nearly constant. After rotation of the input variables, this method can allow significant dimension reduction. Below is a brief summary of the process.

1. Compute the gradient of the quantity of interest, $q = f(\mathbf{x})$, at several locations sampled from the full input space,

$$\nabla_{\mathbf{x}} f_i = \nabla f(\mathbf{x}_i).$$

2. Compute the eigendecomposition of the matrix $\hat{\mathbf{C}}$,

$$\hat{\mathbf{C}} = \frac{1}{M} \sum_{i=1}^M \nabla_{\mathbf{x}} f_i \nabla_{\mathbf{x}} f_i^T = \hat{\mathbf{W}} \hat{\mathbf{\Lambda}} \hat{\mathbf{W}}^T,$$

where $\hat{\mathbf{W}}$ has eigenvectors as columns, $\hat{\mathbf{\Lambda}} = \text{diag}(\hat{\lambda}_1, \dots, \hat{\lambda}_N)$ contains eigenvalues, and N is the total number of parameters.

3. Using a [truncation_method](#) or specifying a [dimension](#) to estimate the active subspace size, split the eigenvectors into active and inactive directions,

$$\hat{\mathbf{W}} = [\hat{\mathbf{W}}_1 \quad \hat{\mathbf{W}}_2].$$

These eigenvectors are used to rotate the input variables.

4. Next the input variables, \mathbf{x} , are expanded in terms of active and inactive variables,

$$\mathbf{x} = \hat{\mathbf{W}}_1 \mathbf{y} + \hat{\mathbf{W}}_2 \mathbf{z}.$$

5. A surrogate is then built as a function of the active variables,

$$g(\mathbf{y}) \approx f(\mathbf{x})$$

For additional information, see:

1. Constantine, Paul G. "Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies". Vol. 2. SIAM, 2015.
2. Constantine, Paul G., Eric Dow, and Qiqi Wang. "Active subspace methods in theory and practice: Applications to kriging surfaces." SIAM Journal on Scientific Computing 36.4 (2014): A1500-A1524.
3. Constantine, Paul, and David Gleich. "Computing Active Subspaces." arXiv preprint arXiv:1408.0545 (2014).

7.3.5.1 truth_model_pointer

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truth_model_pointer](#)

Pointer to specify a full-space model, from which to construct a lower dimensional surrogate

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: actual_model_pointer

Argument(s): STRING

Description

This must point to a model block, identified by [id_model](#). That model will be run to generate gradient data, from which an active subspace model will be identified and built.

See [block_pointer](#) for details about pointers.

7.3.5.2 initial_samples

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [initial_samples](#)

Initial number of samples for sampling-based methods

Specification

Alias: none

Argument(s): INTEGER

Default: model-dependent

Description

The `initial_samples` keyword is used to define the number of initial samples (i.e., randomly chosen sets of variable values) at which to execute a model. The initial samples may later be augmented in an iterative process.

Default Behavior

By default, Dakota will use the minimum number of samples required by the chosen method.

Usage Tips

To obtain linear sensitivities or to construct a linear response surface, at least $\text{dim}+1$ samples should be used, where "dim" is the number of variables. For sensitivities to quadratic terms or quadratic response surfaces, at least $(\text{dim}+1)(\text{dim}+2)/2$ samples are needed. For uncertainty quantification, we recommend at least $10*\text{dim}$ samples. For `variance_based_decomp`, we recommend hundreds to thousands of samples. Note that for `variance_based_decomp`, the number of simulations performed will be $N*(\text{dim}+2)$.

Examples

```
method
  sampling
    sample_type random
    initial_samples = 20
    refinement_samples = 5
```

7.3.5.3 sample_type

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [sample_type](#)

Selection of sampling strategy

Specification

Alias: none

Argument(s): none

Default: random

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Sample Type (Group 1)	lhs	Uses Latin Hypercube Sampling (LHS) to sample variables
			random	Uses purely random Monte Carlo sampling to sample variables

Description

The `sample_type` keyword allows the user to select between two types of sampling: Monte Carlo (pure random) and Latin hypercube (stratified) sampling.

The incremental keywords are deprecated; instead use `samples` together with `refinement_samples`.

Default Behavior

If the `sample_type` keyword is present, it must be accompanied by `lhs` or `random`. In most contexts, `lhs` is the default (exception: `multilevel_sampling` uses Monte Carlo by default).

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
    seed = 83921
```

7.3.5.3.1 lhs

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [sample_type](#)
- [lhs](#)

Uses Latin Hypercube Sampling (LHS) to sample variables

Specification

Alias: none

Argument(s): none

Description

The `lhs` keyword invokes Latin Hypercube Sampling as the means of drawing samples of uncertain variables according to their probability distributions. This is a stratified, space-filling approach that selects variable values from a set of equi-probable bins.

Default Behavior

Latin Hypercube Sampling is the default sampling mode in most contexts (exception: `multilevel_sampling`). To explicitly specify LHS in the Dakota input file, the `lhs` keyword must appear in conjunction with the `sample_type` keyword.

Usage Tips

Latin Hypercube Sampling is very robust and can be applied to any problem. It is fairly effective at estimating the mean of model responses and linear correlations with a reasonably small number of samples relative to the number of variables.

Examples

```
method
  sampling
    sample_type lhs
    samples = 20
```

7.3.5.3.2 random

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [sample_type](#)
- [random](#)

Uses purely random Monte Carlo sampling to sample variables

Specification

Alias: none

Argument(s): none

Description

The `random` keyword invokes Monte Carlo sampling as the means of drawing samples of uncertain variables according to their probability distributions.

Default Behavior

In most contexts, Monte Carlo sampling is not the default sampling mode (exception: `multilevel_sampling`). To change this behavior, the `random` keyword must be specified in conjunction with the `sample_type` keyword.

Usage Tips

Monte Carlo sampling is more computationally expensive than Latin Hypercube Sampling as it requires a larger number of samples to accurately estimate statistics.

Examples

```
method
  sampling
    sample_type random
    samples = 200
```

7.3.5.4 truncation_method

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)

Metric that estimates active subspace size

Specification

Alias: none

Argument(s): none

Default: constantine

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		bing_li	Use the Bing Li "ladle" diagnostic to truncate subspace
	Optional		constantine	Use the Constantine diagnostic to truncate subspace
	Optional		energy	Truncate the subspace based on eigenvalue energy
	Optional		cross_validation	Truncate the subspace to minimize surrogate cross-validation error

Description

Metric that controls how many basis vectors are retained in the active subspace.

Default Behavior

The default is to use the [constantine](#) diagnostic.

Usage Tips

If the automated subspace identification methods do not yield desirable results, consider using the explicit [dimension](#) truncation option.

7.3.5.4.1 [bing_li](#)

- [Keywords Area](#)
- [model](#)

- [active_subspace](#)
- [truncation_method](#)
- [bing_li](#)

Use the Bing Li "ladle" diagnostic to truncate subspace

Specification

Alias: none

Argument(s): none

Description

Uses a trade-off criterion to determine where to truncate the active subspace. The criterion is a function of the eigenvalues and eigenvectors of the active subspace gradient matrix. This function compares the decrease in eigenvalue amplitude with the increase in eigenvector variability under bootstrap sampling of the gradient matrix. The active subspace size is taken to be the index of the first minimum of this quantity.

Usage Tips

If this automated diagnostic does not yield desirable results, consider using the explicit [dimension](#) truncation option or one of the other truncation methods.

Theory

Below is a brief outline of the Bing Li method of active subspace identification. The first two steps are common to all active subspace truncation methods.

1. Compute the gradient of the quantity of interest, $q = f(\mathbf{x})$, at several locations sampled from the input space,

$$\nabla_{\mathbf{x}} f_i = \nabla f(\mathbf{x}_i).$$

2. Compute the eigendecomposition of the matrix $\hat{\mathbf{C}}$,

$$\hat{\mathbf{C}} = \frac{1}{M} \sum_{i=1}^M \nabla_{\mathbf{x}} f_i \nabla_{\mathbf{x}} f_i^T = \hat{\mathbf{W}} \hat{\mathbf{\Lambda}} \hat{\mathbf{W}}^T,$$

where $\hat{\mathbf{W}}$ has eigenvectors as columns, $\hat{\mathbf{\Lambda}} = \text{diag}(\hat{\lambda}_1, \dots, \hat{\lambda}_N)$ contains eigenvalues, and N is the total number of parameters.

3. Normalize the eigenvalues,

$$\lambda_i = \frac{\hat{\lambda}_i}{\sum_j^N \hat{\lambda}_j}.$$

4. Use bootstrap sampling of the gradients found in step 1 to compute replicate eigendecompositions,

$$\hat{\mathbf{C}}_j^* = \hat{\mathbf{W}}_j^* \hat{\mathbf{\Lambda}}_j^* (\hat{\mathbf{W}}_j^*)^T.$$

5. Compute variability of eigenvectors,

$$f_i^0 = \frac{1}{M_{boot}} \sum_j^{M_{boot}} \{1 - |\det(\hat{\mathbf{W}}_i^T \hat{\mathbf{W}}_{j,i}^*)|\},$$

where $\hat{\mathbf{W}}_i$ and $\hat{\mathbf{W}}_{j,i}^*$ both contain only the first i eigenvectors and M_{boot} is the number of bootstrap samples. The value of the variability at the first index, f_1^0 , is defined as zero.

6. Normalize the eigenvector variability,

$$f_i = \frac{f_i^0}{\sum_j^N f_j^0}.$$

7. The criterion, g_i , is defined as,

$$g_i = \lambda_i + f_i.$$

8. The index of first minimum of g_i is then the estimated active subspace rank.

For additional information, see Luo, Wei, and Bing Li. "Combining eigenvalues and variation of eigenvectors for order determination." SIAM, 2015.

7.3.5.4.2 constantine

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [constantine](#)

Use the Constantine diagnostic to truncate subspace

Specification

Alias: none

Argument(s): none

Description

Uses a criterion based on the variability of the subspace estimate. Eigenvectors are computed for bootstrap samples of the gradient matrix. The subspace size associated with the minimum distance between bootstrap eigenvectors and the nominal eigenvectors is the estimated active subspace size.

Usage Tips

If this automated diagnostic does not yield desirable results, consider using the explicit [dimension](#) truncation option or one of the other truncation methods.

Theory

Below is a brief outline of the Constantine method of active subspace identification. The first two steps are common to all active subspace truncation methods.

1. Compute the gradient of the quantity of interest, $q = f(\mathbf{x})$, at several locations sampled from the input space,

$$\nabla_{\mathbf{x}} f_i = \nabla f(\mathbf{x}_i).$$

2. Compute the eigendecomposition of the matrix $\hat{\mathbf{C}}$,

$$\hat{\mathbf{C}} = \frac{1}{M} \sum_{i=1}^M \nabla_{\mathbf{x}} f_i \nabla_{\mathbf{x}} f_i^T = \hat{\mathbf{W}} \hat{\mathbf{\Lambda}} \hat{\mathbf{W}}^T,$$

where $\hat{\mathbf{W}}$ has eigenvectors as columns, $\hat{\mathbf{\Lambda}} = \text{diag}(\hat{\lambda}_1, \dots, \hat{\lambda}_N)$ contains eigenvalues, and N is the total number of parameters.

3. Use bootstrap sampling of the gradients found in step 1 to compute replicate eigendecompositions,

$$\hat{\mathbf{C}}_j^* = \hat{\mathbf{W}}_{j,n}^* (\hat{\mathbf{W}}_j^*)^T.$$

4. Compute the average distance between nominal and bootstrap subspaces,

$$e_n^* = \frac{1}{M_{boot}} \sum_j^{M_{boot}} \text{dist}(\text{ran}(\hat{\mathbf{W}}_n), \text{ran}(\hat{\mathbf{W}}_{j,n}^*)) = \frac{1}{M_{boot}} \sum_j^{M_{boot}} \left\| \hat{\mathbf{W}}_n \hat{\mathbf{W}}_n^T - \hat{\mathbf{W}}_{j,n}^* (\hat{\mathbf{W}}_{j,n}^*)^T \right\|,$$

where M_{boot} is the number of bootstrap samples, $\hat{\mathbf{W}}_n$ and $\hat{\mathbf{W}}_{j,n}^*$ both contain only the first n eigenvectors, and $n < N$.

5. The estimated subspace rank, r , is then,

$$r = \arg \min_n e_n^*.$$

For additional information, see Constantine, Paul G. "Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies". Vol. 2. SIAM, 2015.

7.3.5.4.3 energy

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [energy](#)

Truncate the subspace based on eigenvalue energy

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		truncation_tolerance	Specify the maximum percentage (as a decimal) of the eigenvalue energy not captured by the active subspace representation.

Description

Uses a criterion based on the derivative matrix eigenvalue energy.

Usage Tips

This subspace truncation method may work best when working with non-normally distributed uncertain variables. If this automated diagnostic does not yield desirable results, consider using the explicit [dimension](#) truncation option or one of the other truncation methods.

Theory

Using the eigenvalue energy truncation metric, the subspace size is determined using the following equation:

$$n = \inf \left\{ d \in \mathbf{Z} \mid 1 \leq d \leq N \wedge 1 - \frac{\sum_{i=1}^d \lambda_i}{\sum_{i=1}^N \lambda_i} < \varepsilon \right\}$$

where ε is the `truncation_tolerance`, n is the estimated subspace size, N is the size of the full space, and λ_i are the eigenvalues of the derivative matrix.

7.3.5.4.3.1 truncation_tolerance

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [energy](#)
- [truncation_tolerance](#)

Specify the maximum percentage (as a decimal) of the eigenvalue energy not captured by the active subspace representation.

Specification

Alias: none

Argument(s): REAL

Description

The `truncation_tolerance` is used in the energy truncation method to identify an active subspace. The `truncation_tolerance` is used in the following equation to determine the subspace size:

$$n = \inf \left\{ d \in \mathbf{Z} \mid 1 \leq d \leq N \wedge 1 - \frac{\sum_{i=1}^d \lambda_i}{\sum_{i=1}^N \lambda_i} < \varepsilon \right\}$$

where ε is the `truncation_tolerance`, n is the estimated subspace size, N is the size of the full space, and λ_i are the eigenvalues of the derivative matrix.

The default value for `truncation_tolerance` is 1×10^{-6} .

7.3.5.4.4 cross_validation

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [cross_validation](#)

Truncate the subspace to minimize surrogate cross-validation error

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	CV Selection Criterion (Group 1)	minimum	Select subspace to minimize cross-validation error
			relative	Choose subspace with cross-validation error less than tolerance
			decrease	Choose subspace where cross-validation error stabilizes
	Optional		relative_tolerance	Tolerance for cross-validation error value
	Optional		decrease_ - tolerance	Tolerance for cross-validation error stabilization
	Optional		max_rank	Maximum subspace dimension to consider
	Optional		exhaustive	Assess all admissible subspace dimensions

Description

Select the subspace dimension that minimizes the 10-fold cross-validation error when constructing a moving least squares surrogate model.

7.3.5.4.4.1 minimum

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)

- [cross_validation](#)
- [minimum](#)

Select subspace to minimize cross-validation error

Specification

Alias: none

Argument(s): none

Description

Select the subspace dimension that mimimizes the cross-validation error across sizes considered.

7.3.5.4.4.2 relative

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [cross_validation](#)
- [relative](#)

Choose subspace with cross-validation error less than tolerance

Specification

Alias: none

Argument(s): none

Description

Select smallest subspace with cross-validation error less than specified tolerance.

7.3.5.4.4.3 decrease

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [cross_validation](#)
- [decrease](#)

Choose subspace where cross-validation error stabilizes

Specification

Alias: none

Argument(s): none

Description

Select smallest subspace where change in cross-validation error to next larger subspace falls below specified tolerance.

7.3.5.4.4.4 relative_tolerance

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [cross_validation](#)
- [relative_tolerance](#)

Tolerance for cross-validation error value

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-6

Description

Select smallest subspace where change in cross-validation error falls below specified tolerance.

7.3.5.4.4.5 decrease_tolerance

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [cross_validation](#)
- [decrease_tolerance](#)

Tolerance for cross-validation error stabilization

Specification

Alias: none

Argument(s): REAL

Default: 1.0e-6

Description

Select smallest subspace where change in cross-validation error falls below specified tolerance.

7.3.5.4.4.6 max_rank

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [cross_validation](#)
- [max_rank](#)

Maximum subspace dimension to consider

Specification

Alias: none

Argument(s): INTEGER

Default: number fullspace vars

Description

In cross-validation, only consider subspace dimensions from 1 to `max_rank`.

7.3.5.4.4.7 exhaustive

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [truncation_method](#)
- [cross_validation](#)
- [exhaustive](#)

Assess all admissible subspace dimensions

Specification

Alias: none

Argument(s): none

Default: on

Description

Assess cross-validation error for all admissible subspace dimensions, even if a particular dimension meets specified cross-validation tolerance(s).

7.3.5.5 dimension

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [dimension](#)

Explicitly specify the desired subspace size

Specification

Alias: none

Argument(s): INTEGER

Description

This control explicitly indicates the number of basis vectors to retain. The subspace model will include exactly `dimension` variables.

Default Behavior

Active for adapted basis. Not active for active subspace (the number of basis vectors will be chosen by one of the truncation methods).

Usage Tips

This control can be helpful when *a priori* studies give insight to the appropriate subspace size.

7.3.5.6 bootstrap_samples

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [bootstrap_samples](#)

Number of bootstrap replicates used in truncation metrics

Specification

Alias: none

Argument(s): INTEGER

Description

The number of bootstrap replicates used to estimate the active subspace size.

Default Behavior

Use 100 replicates.

7.3.5.7 build_surrogate

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [build_surrogate](#)

Construct moving least squares surrogate over active subspace

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		refinement_ - samples	Number of supplementary surrogate build samples

Description

Once the active subspace variables have been identified, replace the fullspace truth model with a moving least squares surrogate model over the reduced variables. The surrogate is constructed using the fullspace function value samples collected during subspace identification, possibly supplemented with additional [refinement_samples](#) in the fullspace.

Default Behavior

No surrogate is constructed by default.

Examples

```
model
  active_subspace
    id_model = 'SUBSPACE'
    truth_model_pointer = 'FULLSPACE'
    initial_samples 100
  build_surrogate
    refinement_samples 10
```

7.3.5.7.1 refinement_samples

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [build_surrogate](#)
- [refinement_samples](#)

Number of supplementary surrogate build samples

Specification

Alias: none

Argument(s): INTEGERLIST

Default: 0

Description

Augment the function value data collected during subspace identification with the specified number of `refinement_samples` to build the surrogate model.

7.3.5.8 normalization

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [normalization](#)

Normalize gradient samples

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Normalize By... (Group 1)	mean_value	Normalize by sample mean of function values
			mean_gradient	Normalize by sample mean of gradient norms
			local_gradient	Normalize each gradient sample by its norm

Description

Normalize the matrix of sampled gradients before identifying subspace.

Default Behavior

The default is [local_gradient](#)

7.3.5.8.1 mean_value

- [Keywords Area](#)

- [model](#)
- [active_subspace](#)
- [normalization](#)
- [mean_value](#)

Normalize by sample mean of function values

Specification

Alias: none

Argument(s): none

Description

For each reponse function f_i , normalize its gradient data by the mean m_i of the function values taken across samples $j = 1, \dots, J$:

$$m_i = \frac{1}{J} \sum_{j=1}^J f_i^j$$

7.3.5.8.2 mean_gradient

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [normalization](#)
- [mean_gradient](#)

Normalize by sample mean of gradient norms

Specification

Alias: none

Argument(s): none

Description

For each reponse function f_i , normalize its gradient data by the mean m_i of the gradient two-norms taken across samples $j = 1, \dots, J$:

$$m_i = \frac{1}{J} \sum_{j=1}^J \|\nabla^j f_i\|_2$$

7.3.5.8.3 local_gradient

- [Keywords Area](#)
- [model](#)
- [active_subspace](#)
- [normalization](#)
- [local_gradient](#)

Normalize each gradient sample by its norm

Specification

Alias: none

Argument(s): none

Description

For each response function f_i , normalize each gradient sample $\nabla^j f_i$ by its two-norm $\|\nabla^j f_i\|$

7.3.6 adapted_basis

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)

Basis adaptation model

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		truth_model_pointer	Pointer to specify a "truth" model, from which to construct a surrogate
	Optional		truncation_tolerance	Convergence tolerance used to identify the (reduced) dimension of the rotation matrix

Description

A model that transforms the original model (given by `truth_model_pointer`) to one with a rotated set of variables. The current implementation does not support the reduction of the dimension for the new model and only transform the variable set in a rotated one in which the variables are arranged in a decreasing importance order.

An initial PCE representation is built with a sparse grid method and the PCE coefficients are then used to build a rotation matrix. This matrix is also reported as a result of the code execution. The new model can subsequently be used for an UQ workflow.

Expected Output A basis adaptation model will perform an initial sparse grid design to identify the rotation matrix.

Examples

Perform an initial sparse grid design (level 1) to evaluate the PCE expansion and evaluate the rotation matrix. Afterwards, 100 samples are generated for the model in the rotated space to obtain its statistics.

```
method,
  sampling
    model_pointer = 'SUBSPACE'
    samples = 100 seed = 1234567

model
  id_model = 'SUBSPACE'
  adapted_basis
    truth_model_pointer = 'FULLSPACE'
    sparse_grid_level = 1
```

Theory

The idea behind the Basis Adaptation method is to generate a PCE representation and rotate it such that the new set of rotated variables are organized in a decreasing importance ordering. Subsequently this rotation matrix can be truncated according to some criterion to only include the significant directions.

The first step of the Basis Adaptation is to compute a PCE expansion (we assume here ξ to have a standard multivariate Gaussian distribution)

$$Q(\xi) = \sum_{\alpha \in \mathcal{J}_{d,p}} Q_{\alpha} \psi_{\alpha}(\xi),$$

where $\alpha = (\alpha_1, \dots, \alpha_d) \in \mathcal{J}_{d,p} := (\mathbb{N}_0)^d$ with $|\alpha| = \sum_{i=1}^d \alpha_i \leq d$ is multi-index of dimension d and order up to p .

Afterwards, a multivariate Gaussian distribution η is sought such that

$$\eta = A\xi,$$

where A is an isometry such that $AA^T = I$.

The basis adaptation model is obtained by expressing the original model with respect to the rotated set of variables η as

$$Q^A(\eta) = \sum_{\beta \in \mathcal{J}_{d,p}} Q_{\beta}^A \psi_{\beta}(\eta).$$

Since the basis for both the original and the adapted basis model span the same space, we know that $Q^A(\eta(\xi)) = Q(\xi)$ and therefore a relationships between the PCE coefficients exists

$$Q_{\alpha} = \sum_{\beta \in \mathcal{J}_{d,p}} Q_{\beta}^A \langle \psi_{\beta}^A, \psi_{\alpha} \rangle, \quad \alpha \in \mathcal{J}_{d,p}.$$

The linear adaptation strategy is used at this time in Dakota to obtain the rotation matrix A . The steps to obtain the matrix A are the following:

1. The first variable η_1 is obtained by directly combining the first order terms, that is

$$\eta_1 = \sum_{\alpha \in \mathcal{J}_{d,1}} Q_\alpha \psi_\alpha(\xi) = \sum_{i=1}^d Q_{e_i} \xi_i,$$

where e_i is the d -dimensional multi-index with 1 at the i th location and zero elsewhere.

2. The remaining rows of the A matrix ($A \in \mathbb{R}^{d \times d}$) are obtained by using the largest coefficient, row after row, placed in the column that it occupies in the first row. All other terms are set to zero.
3. As last step, in order to obtain an isometry, the Gram-Schmidt procedure is followed on A .

See the theory manual for more details.

See Also

These keywords may also be of interest:

- [active_subspace](#)

7.3.6.1 truth_model_pointer

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truth_model_pointer](#)

Pointer to specify a "truth" model, from which to construct a surrogate

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: actual_model_pointer

Argument(s): STRING

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Expansion Basis Control (Group 1)	sparse_grid_level	Level to use in sparse grid integration or interpolation

		expansion_order	The (initial) order of a polynomial expansion
	Optional	dimension	Prescribe the number of active dimensions
	Optional	rotation_method	Method used to build the rotation matrix

Description

This must point to a model block, identified by [id_model](#). That model will be run to generate training data, from which a surrogate model will be constructed.

See [block_pointer](#) for details about pointers.

7.3.6.1.1 [sparse_grid_level](#)

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truth_model_pointer](#)
- [sparse_grid_level](#)

Level to use in sparse grid integration or interpolation

Specification

Alias: none

Argument(s): INTEGER

Description

Multi-dimensional integration by the Smolyak sparse grid method (specified with [sparse_grid_level](#) and, optionally, [dimension_preference](#)). The underlying one-dimensional integration rules are the same as for the tensor-product quadrature case; however, the default rule selection is nested for sparse grids (Genz-Keister for normals/transformed normals and Gauss-Patterson for uniforms/transformed uniforms). This default can be overridden with an explicit [non_nested](#) specification (resulting in Gauss-Hermite for normals/transformed normals and Gauss-Legendre for uniforms/transformed uniforms). As for tensor quadrature, the [dimension_preference](#) specification enables the use of anisotropic sparse grids (refer to the PCE description in the User's Manual for the anisotropic index set constraint definition). Similar to anisotropic tensor grids, the dimension with greatest preference will have resolution at the full [sparse_grid_level](#) and all other dimension resolutions will be reduced in proportion to their reduced preference. For PCE with either isotropic or anisotropic sparse grids, a summation of tensor-product expansions is used, where each anisotropic tensor-product quadrature rule underlying the sparse grid construction results in its own anisotropic tensor-product expansion as described in case 1. These anisotropic tensor-product expansions are summed into a sparse PCE using the standard Smolyak summation (again, refer to the User's Manual for additional details). As for [quadrature_order](#), the [sparse_grid_level](#) specification admits an array input for enabling specification of multiple grid resolutions used by certain advanced solution methodologies.

A corresponding sequence specification is documented at, e.g., [sparse_grid_level_sequence](#) and [sparse_grid_level_sequence](#)

See Also

These keywords may also be of interest:

- [sparse_grid_level_sequence](#) method-multifidelity_stoch_collocation-sparse_grid_level_sequence

7.3.6.1.2 expansion_order

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truth_model_pointer](#)
- [expansion_order](#)

The (initial) order of a polynomial expansion

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		collocation_ratio	Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Description

When the `expansion_order` for a polynomial chaos expansion is specified, the coefficients may be computed by integration based on random samples or by regression using either random or sub-sampled tensor product quadrature points.

Multidimensional integration by Latin hypercube sampling (specified with `expansion_samples`). In this case, the expansion order p cannot be inferred from the numerical integration specification and it is necessary to provide an `expansion_order` to specify p for a total-order expansion.

Linear regression (specified with either `collocation_points` or `collocation_ratio`). A total-order expansion is used and must be specified using `expansion_order` as described in the previous option. To avoid requiring the user to calculate N from n and p , the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces `samples = 2N`). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$). When admissible, a constrained least squares approach is employed in which response values are first reproduced exactly and error in reproducing response derivatives is minimized. Two collocation grid options are supported: the default is Latin hypercube sampling

("point collocation"), and an alternate approach of "probabilistic collocation" is also available through inclusion of the `tensor_grid` keyword. In this alternate case, the collocation grid is defined using a subset of tensor-product quadrature points: the order of the tensor-product grid is selected as one more than the expansion order in each dimension (to avoid sampling at roots of the basis polynomials) and then the tensor multi-index is uniformly sampled to generate a non-repeated subset of tensor quadrature points.

If `collocation_points` or `collocation_ratio` is specified, the PCE coefficients will be determined by regression. If no regression specification is provided, appropriate defaults are defined. Specifically SVD-based least-squares will be used for solving over-determined systems and under-determined systems will be solved using LASSO. For the situation when the number of function values is smaller than the number of terms in a PCE, but the total number of samples including gradient values is greater than the number of terms, the resulting over-determined system will be solved using equality constrained least squares. Technical information on the various methods listed below can be found in the Linear regression section of the Theory Manual. Some of the regression methods (OMP, LASSO, and LARS) are able to produce a set of possible PCE coefficient vectors (see the Linear regression section in the Theory Manual). If cross validation is inactive, then only one solution, consistent with the `noise_tolerance`, will be returned. If cross validation is active, Dakota will choose between possible coefficient vectors found internally by the regression method across the set of expansion orders (1,...,expansion_order) and the set of specified noise tolerances and return the one with the lowest cross validation error indicator.

7.3.6.1.2.1 collocation_ratio

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truth_model_pointer](#)
- [expansion_order](#)
- [collocation_ratio](#)

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion.

Specification

Alias: none

Argument(s): REAL

Description

Set the number of points used to build a PCE via regression to be proportional to the number of terms in the expansion. To avoid requiring the user to calculate N from n and p, the `collocation_ratio` allows for specification of a constant factor applied to N (e.g., `collocation_ratio = 2.` produces samples = 2N). In addition, the default linear relationship with N can be overridden using a real-valued exponent specified using `ratio_order`. In this case, the number of samples becomes cN^o where c is the `collocation_ratio` and o is the `ratio_order`. The `use_derivatives` flag informs the regression approach to include derivative matching equations (limited to gradients at present) in the least squares solutions, enabling the use of fewer collocation points for a given expansion order and dimension (number of points required becomes $\frac{cN^o}{n+1}$).

7.3.6.1.3 dimension

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)

- [truth_model_pointer](#)
- [dimension](#)

Prescribe the number of active dimensions

Specification

Alias: none

Argument(s): INTEGER

Description

If a prescribed dimension for the rotation matrix is desired, this can be specified with the keyword `dimension`. In this case, the algorithm does not use the `truncation_method` or the `truncation_tolerance` to determine the truncation of the rotation matrix, but it just selects the first d dimensions, where d is assigned by `dimension`.

Examples

The following block

```
model
  id_model = 'SUBSPACE'
  adapted_basis
    truth_model_pointer = 'FULLSPACE'
    sparse_grid_level = 1
    dimension = 3
```

prescribes the selection of the first three most important directions to build the rotation matrix and the reduced model representation.

7.3.6.1.4 rotation_method

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truth_model_pointer](#)
- [rotation_method](#)

Method used to build the rotation matrix

Specification

Alias: none

Argument(s): none

Default: norm

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Rotation Method (Group 1)	unranked	Use the unranked method to obtain the rotation matrix
			ranked	Use the ranked method to obtain the rotation matrix

Description

The rotation matrix for the Adapted Basis method is built starting from a matrix which includes the linear PCE coefficients. The entries of the first row are the linear PCE coefficients. For all the other rows, each row has only one nonzero entry. Depending on the way used to construct all the rows, from the second to last one, there are two implemented methods, [unranked](#) and [ranked](#). A followed Gram_schmidt process is applied on the resulting matrix to make it an isometry.

Default Behavior

The default is to use the [ranked](#).

Usage Tips

Examples

The following method block

```
model
  id_model = 'SUBSPACE'
  adapted_basis
    truth_model_pointer = 'FULLSPACE'
    sparse_grid_level = 1
    rotation_method = unranked
```

changes the default method to the [unranked](#) method.

7.3.6.1.4.1 unranked

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truth_model_pointer](#)
- [rotation_method](#)
- [unranked](#)

Use the unranked method to obtain the rotation matrix

Specification

Alias: none

Argument(s): none

Description

In this case the construction of the second to last rows of the rotation matrix is performed by using the value 1 in the diagonal terms and zeros elsewhere.

7.3.6.1.4.2 ranked

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truth_model_pointer](#)
- [rotation_method](#)
- [ranked](#)

Use the ranked method to obtain the rotation matrix

Specification

Alias: none

Argument(s): none

Description

In this case the construction of the i th row, with $i \geq 2$, of the rotation matrix is performed by putting the $(i - 1)$ th largest linear PCE coefficient in the column that it occupies in the first row.

Theory

The first row of the rotation captures the complete Gaussian components and is suggested as the leading direction. By the construction in this method, the other rotated variables are the variables in the original space that have the greatest sensitivity in descending order. The particular construction makes sure that the importance of the adapted variables is in descending order, which permits the possibility of reduced representation.

7.3.6.2 truncation_tolerance

- [Keywords Area](#)
- [model](#)
- [adapted_basis](#)
- [truncation_tolerance](#)

Convergence tolerance used to identify the (reduced) dimension of the rotation matrix

Specification

Alias: none

Argument(s): REAL

Default: 0.8

Description

This convergence tolerance controls the dimension of the rotation matrix after the truncation. The metric controlled by this convergence tolerance is a quantity varying between [0,1]. If a more aggressive dimension truncation is desired, the default value (0.8) can be reduced to a lower value.

Examples

The following block

```
model
  id_model = 'SUBSPACE'
  adapted_basis
    truth_model_pointer = 'FULLSPACE'
    sparse_grid_level = 1
    truncation_tolerance = 0.9
```

increases the default value (0.8) for the `truncation_tolerance` to 0.9 to obtain a rotation matrix and, consequently, a reduced model that more closely approximates the full dimensional model.

7.3.7 random_field

- [Keywords Area](#)
- [model](#)
- [random_field](#)

Experimental capability to generate a random field representation. from data, from simulation runs, or from a covariance matrix. The representation may then be sampled for use as a random field input to another simulation. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		build_source	Specify how the random field will be built: from a data file, from simulation runs, or from a covariance matrix. THIS IS AN EXPERIMENTAL CAPABILITY.

	Optional	expansion_form	Specify the form of the expansion to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.
	Optional	expansion_bases	Specify the number of basis functions to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.
	Optional	truncation_tolerance	Specify a percent of the response variance that should be captured with the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.
	Required	propagation_model_pointer	Pointer to the model that will accept realizations of the random field and use them for subsequent analysis. Typically, this model will take the random field as inputs, e.g. a random field defining a pressure boundary or temperature boundary condition over a structure. THIS IS AN EXPERIMENTAL CAPABILITY.

Description

Capability to generate a random field representation from data, from simulation runs, or from a covariance matrix. The random field may then be sampled for use as a random field input to another simulation. THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior There are three main sections of the `random_field` model. The first section tells Dakota what data to use to build the random field. This is specified with `build_source`. The source of data to build

the random field may be a file with data (where the N rows of data correspond to N samples of the random field and the M columns correspond to field values), or it may be a simulation that generates field data, or it may be specified given a mesh and a covariance matrix governing how the field varies over the mesh. In the case of using a simulation to generate field data, the simulation is defined with `dace_method_pointer`. In the case of using a mesh and a covariance, the form of the covariance is defined with `analytic_covariance`.

The next section of the random field model specifies the form of the expansion, `expansion_form`. This can be either a Karhunen-Loeve expansion or a Principal components analysis. These are very similar: both involve the eigenvalues of the covariance matrix of the field data. The only difference is in the treatment of the estimation of the coefficients of the eigenvector basis functions. In the PCA case, we have developed an approach which makes the coefficients explicit functions of the uncertain variables used to generate the random field. The specification of the random field can also include the number of bases to retain or a truncation tolerance, which defines the percent variance that the expansion should capture.

The final section of the random field model allows the user to specify a pointer to a model over which the random field will be propagated, meaning the model which will be driven with the random field input. This part of the specification is optional: one can build a random field but not use it in a downstream model.

Examples

As stated above, this is an emerging capability. The syntax currently looks like the following:

```
random_field
  build_source
    rf_data_file | dace_method_pointer | analytic_covariance
  expansion_form
    karhunen_loeve | principal_components
  expansion_bases
  truncation_tolerance
  propagation_model_pointer
```

7.3.7.1 build_source

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [build_source](#)

Specify how the random field will be built: from a data file, from simulation runs, or from a covariance matrix. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Build Data Source (Group 1)	rf_data_file	Specify that the random field will be built from a file of data. THIS IS AN EXPERIMENTAL CAPABILITY.
			dace_method_pointer	Pointer to a DACE method for purposes of generating an ensemble of field responses to be used in estimating a random field model. THIS IS AN EXPERIMENTAL CAPABILITY.
			analytic_covariance	Use an analytic covariance function for the purposes of generating a random field model. THIS IS AN EXPERIMENTAL CAPABILITY.

Description

As part of the capability to generate a random field representation, the user needs to specify the data used to generate the random field representation. This data may reside in a data file, it may be generated by running a set of simulations and generating field responses, or it may be generated by a covariance matrix defined over a mesh. THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior Currently, the `build_source` that is fully working is the `dace_method_pointer`. The others are not fully operational.

7.3.7.1.1 `rf_data_file`

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [build_source](#)
- [rf_data_file](#)

Specify that the random field will be built from a file of data. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): STRING

Default: none

Description

As part of the capability to generate a random field representation, the user needs to specify the data used to generate the random field representation. In the case of `rf_data_file`, the data should reside in that file. The rows of this file represent separate samples, and the columns represent the field data. For example, if you have 100 samples of a random field of length 500, the data file will be of dimension 100 x 500. Currently, this option is not operational. THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior Currently, the `build_source` that is fully working is the `dace_method_pointer`. The others are not fully operational.

7.3.7.1.2 `dace_method_pointer`

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [build_source](#)
- [dace_method_pointer](#)

Pointer to a DACE method for purposes of generating an ensemble of field responses to be used in estimating a random field model. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): STRING

Default: no design of experiments data

Description

As part of the capability to generate a random field representation, the user needs to specify the data used to generate the random field representation. One way to do this is to run a set of simulations and generating field responses. Dakota will then take the full set of field responses (e.g. multiple samples, where each sample has a field response) and construct a random field model representing the uncertainty in the ensemble. The `dace_method_pointer` is a pointer to a Design and Analysis of Computer Experiments (DACE) method, typically which is a sampling method. In this case, the sampling method should be on a simulation which can generate field responses (e.g. `field_responses`, `field_objectives`, or `field_calibration` terms.)

THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior Currently, the `build_source` that is fully working is the `dace_method_pointer`. The others are not fully operational.

7.3.7.1.3 `analytic_covariance`

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [build_source](#)

- [analytic_covariance](#)

Use an analytic covariance function for the purposes of generating a random field model. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Covariance Kernel Form (Group 1)	Dakota Keyword squared_exponential	Dakota Keyword Description Specify a squared exponential covariance in the case where the random field is built from an analytic covariance functio. THIS IS AN EXPERIMENTAL CAPABILITY.
			exponential	Specify an exponential covariance in the case where the random field is built from an analytic covariance functio. THIS IS AN EXPERIMENTAL CAPABILITY.

Description

As part of the capability to generate a random field representation, the user needs to specify the data used to generate the random field representation. If `analytic_covariance` is specified, an analytic covariance function will be used to generate instantiations of a random field over a mesh. The form of the covariance function must be specified (e.g. `exponential` or `squared_exponential`). THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior Currently, the `build_source` that is fully working is the `dace_method_pointer`. The others are not fully operational.

7.3.7.1.3.1 `squared_exponential`

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [build_source](#)
- [analytic_covariance](#)

- [squared_exponential](#)

Specify a squared exponential covariance in the case where the random field is built from an analytic covariance function. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Description

THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior Currently, the `build_source` that is fully working is the `dace_method_pointer`. The others are not fully operational.

7.3.7.1.3.2 exponential

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [build_source](#)
- [analytic_covariance](#)
- [exponential](#)

Specify an exponential covariance in the case where the random field is built from an analytic covariance function. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Description

THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior Currently, the `build_source` that is fully working is the `dace_method_pointer`. The others are not fully operational.

7.3.7.2 expansion_form

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [expansion_form](#)

Specify the form of the expansion to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Reduced Basis Type (Group 1)	karhunen_loeve	Specify Karhunen-Loeve as the expansion form to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.
			principal_components	Specify Principal Components as the form of the expansion to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.

Description

This control allows the user to specify the form of the random field representation (e.g. either a Karhunen-Loeve or Principal Components expansion). THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior The default expansion form is Karhunen-Loeve.

7.3.7.2.1 karhunen_loeve

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [expansion_form](#)
- [karhunen_loeve](#)

Specify Karhunen-Loeve as the expansion form to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Description

THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior The default expansion form is Karhunen-Loeve.

7.3.7.2.2 principal_components

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [expansion_form](#)
- [principal_components](#)

Specify Principal Components as the form of the expansion to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): none

Description

Principal Components and Karhunen-Loeve are very similar functional forms of the expansion. They both use basis functions which are eigenvectors of the covariance matrix of the random field data. However, in principal components, we parameterize the coefficients of the expansion to be Gaussian process models which are functions of the uncertain parameters used to generate the initial ensemble of data representing the random field. THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior The default expansion form is Karhunen-Loeve.

7.3.7.3 expansion_bases

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [expansion_bases](#)

Specify the number of basis functions to be used in the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): INTEGER

Description

This control allows the user to specify the number of basis functions to be used in the random field representation (e.g. either a Karhunen-Loeve or Principal Components expansion). Typically, a small number of basis functions (3-5) will be sufficient to represent a significant amount of the variance in the response field. However, this depends on the particulars of the problem. THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior If the user specifies neither `trunction_tolerance` nor `expansion_bases`, then the number of expansion bases that captures 95% of the variance will be used.

7.3.7.4 `truncation_tolerance`

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [truncation_tolerance](#)

Specify a percent of the response variance that should be captured with the random field representation. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): REAL

Description

This control allows the user to specify the percent variance to be capture in the random field representation (e.g. either a Karhunen-Loeve or Principal Components expansion). Typically, the user would specify something like 0.9 or 0.95, where 0.9 means that 90% of the variance of the response field will be captured by the random field. THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior If the user specifies neither `trunction_tolerance` nor `expansion_bases`, then the truncation tolerance is set to 0.95 and the number of expansion bases that captures 95% of the variance will be used.

7.3.7.5 `propagation_model_pointer`

- [Keywords Area](#)
- [model](#)
- [random_field](#)
- [propagation_model_pointer](#)

Pointer to the model that will accept realizations of the random field and use them for subsequent analysis. Typcially, this model will take the random field as inputs, e.g. a random field defining a pressure boundary or temperature boundary condition over a structure. THIS IS AN EXPERIMENTAL CAPABILITY.

Specification

Alias: none

Argument(s): STRING

Description

After a random field representation has been generated, Dakota will generate samples of that random field based on the representation. These sample realizations can then be used to drive another set of simulation analyses. The `propagation_model_pointer` is the model on which the random field realizations will be propagated. THIS IS AN EXPERIMENTAL CAPABILITY UNDER ACTIVE DEVELOPMENT.

Default Behavior

7.3.8 `variables_pointer`

- [Keywords Area](#)
- [model](#)
- [variables_pointer](#)

Specify which variables block will be included with this model block

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: model use of last variables parsed

Description

The `variables_pointer` is used to specify which variables block will be used by the model, by cross-referencing with `id_variables` keyword in the `variables` block.

See [block_pointer](#) for details about pointers.

Default Behavior

When a variables pointer is not specified, the model will use the last variables block parsed from the input file.

7.3.9 `responses_pointer`

- [Keywords Area](#)
- [model](#)
- [responses_pointer](#)

Specify which reponses block will be used by this model block

Topics

This keyword is related to the topics:

- [block_pointer](#)

Specification

Alias: none

Argument(s): STRING

Default: model use of last responses parsed

Description

The `responses_pointer` is used to specify which responses block will be used by the model, by cross-referencing with `id_responses` keyword in the `responses` block.

See [block_pointer](#) for details about pointers.

Default Behavior

When a responses pointer is not specified, the model will use the last responses block parsed from the input file.

7.3.10 hierarchical_tagging

- [Keywords Area](#)
- [model](#)
- [hierarchical_tagging](#)

Enables hierarchical evaluation tagging

Specification

Alias: none

Argument(s): none

Default: no hierarchical tagging

Description

The hierarchical tagging option is useful for studies involving multiple models with a nested or hierarchical relationship. For example a nested model has a sub-method, which itself likely operates on a sub-model, or a hierarchical approximation involves coordination of low and high fidelity models. Specifying `hierarchical_tagging` will yield function evaluation identifiers ("tags") composed of the evaluation IDs of the models involved, e.g., `outermodel.-innermodel.interfaceid = 4.9.2`. This communicates the outer contexts to the analysis driver when performing a function evaluation.

Examples

```
test/dakota_uq_timeseries_ivp_optinterf.in test/dakota_uq_timeseries_sop_optinterf.in
```

See Also

These keywords may also be of interest:

- [file_tag](#) model-nested

7.4 variables

- [Keywords Area](#)
- [variables](#)

Specifies the parameter set to be iterated by a particular method.

Topics

This keyword is related to the topics:

- [block](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		id_variables	Name the variables block; helpful when there are multiple
	Optional		active	Set the active variables view a method will see
	Optional (<i>Choose One</i>)	Variable Domain (Group 1)	mixed	Maintain continuous/discrete variable distinction
			relaxed	Allow treatment of discrete variables as continuous
	Optional		continuous_design	Design variable - continuous
	Optional		discrete_design_range	Design variable - discrete range-valued
	Optional		discrete_design_set	Design variable - discrete set-valued
	Optional		normal_uncertain	Aleatory uncertain variable - normal (Gaussian)

	Optional	lognormal_ - uncertain	Aleatory uncertain variable - lognormal
	Optional	uniform_uncertain	Aleatory uncertain variable - uniform
	Optional	loguniform_ - uncertain	Aleatory uncertain variable - loguniform
	Optional	triangular_ - uncertain	Aleatory uncertain variable - triangular
	Optional	exponential_ - uncertain	Aleatory uncertain variable - exponential
	Optional	beta_uncertain	Aleatory uncertain variable - beta
	Optional	gamma_uncertain	Aleatory uncertain variable - gamma
	Optional	gumbel_uncertain	Aleatory uncertain variable - gumbel
	Optional	frechet_uncertain	Aleatory uncertain variable - Frechet
	Optional	weibull_uncertain	Aleatory uncertain variable - Weibull
	Optional	histogram_bin_ - uncertain	Aleatory uncertain variable - continuous histogram
	Optional	poisson_uncertain	Aleatory uncertain discrete variable - Poisson
	Optional	binomial_uncertain	Aleatory uncertain discrete variable - binomial
	Optional	negative_binomial_ - uncertain	Aleatory uncertain discrete variable - negative binomial
	Optional	geometric_ - uncertain	Aleatory uncertain discrete variable - geometric
	Optional	hypergeometric_ - uncertain	Aleatory uncertain discrete variable - hypergeometric
	Optional	histogram_point_ - uncertain	Aleatory uncertain variable - discrete histogram

	Optional	uncertain_-correlation_matrix	Correlation among aleatory uncertain variables
	Optional	continuous_-interval_uncertain	Epistemic uncertain variable - values from one or more continuous intervals
	Optional	discrete_interval_-uncertain	Epistemic uncertain variable - values from one or more discrete intervals
	Optional	discrete_uncertain_-set	Epistemic uncertain variable - discrete set-valued
	Optional	continuous_state	State variable - continuous
	Optional	discrete_state_-range	State variables - discrete range-valued
	Optional	discrete_state_set	State variable - discrete set-valued
	Optional	linear_inequality_-constraint_matrix	Define coefficients of the linear inequality constraints
	Optional	linear_inequality_-lower_bounds	Define lower bounds for the linear inequality constraint
	Optional	linear_inequality_-upper_bounds	Define upper bounds for the linear inequality constraint
	Optional	linear_inequality_-scale_types	How to scale each linear inequality constraint
	Optional	linear_inequality_-scales	Characteristic values to scale linear inequalities

	Optional	linear_equality_constraint_matrix	Define coefficients of the linear equalities
	Optional	linear_equality_targets	Define target values for the linear equality constraints
	Optional	linear_equality_scale_types	How to scale each linear equality constraint
	Optional	linear_equality_scales	Characteristic values to scale linear equalities

Description

The `variables` specification in a Dakota input file specifies the parameter set to be iterated by a particular method. In the case of

- An optimization study: These variables are adjusted in order to locate an optimal design.
- Parameter studies/sensitivity analysis/design of experiments: These parameters are perturbed to explore the parameter space.
- Uncertainty analysis: The variables are associated with distribution/interval characterizations which are used to compute corresponding distribution/interval characterizations for response functions.

To accommodate these different studies, Dakota supports different:

- Variable types
 - design
 - aleatory uncertain
 - epistemic uncertain
 - state
- Variable domains
 - continuous
 - discrete
 - * discrete range
 - * discrete integer set
 - * discrete string set
 - * discrete real set

Use the [variables](#) page to browse the available variables by type and domain.

Variable Types

- Design Variables
 - Design variables are those variables which are modified for the purposes of seeking an optimal design.
 - The most common type of design variables encountered in engineering applications are of the continuous type. These variables may assume any real value within their bounds.
 - All but a handful of the optimization algorithms in Dakota support continuous design variables exclusively.

- Aleatory Uncertain Variables
 - Aleatory uncertainty is also known as inherent variability, irreducible uncertainty, or randomness.
 - Aleatory uncertainty is predominantly characterized using probability theory. This is the only option implemented in Dakota.
- Epistemic Uncertain Variables
 - Epistemic uncertainty is uncertainty due to lack of knowledge.
 - In Dakota, epistemic uncertainty is assessed by interval analysis or the Dempster-Shafer theory of evidence
 - Continuous or discrete interval or set-valued variables are used to define set-valued probabilities or basic probability assignments (BPA) which define a belief structure.
 - Note that epistemic uncertainty can also be modeled with probability density functions (as done with aleatory uncertainty). Dakota does not support this capability.
- State Variables
 - State variables consist of "other" variables which are to be mapped through the simulation interface, in that they are not to be used for design and they are not modeled as being uncertain.
 - State variables provide a convenient mechanism for managing additional model parameterizations such as mesh density, simulation convergence tolerances, and time step controls.
 - Only parameter studies and design of experiments methods will iterate on state variables.
 - The `initial_value` is used as the only value for the state variable for all other methods, unless `active state` is invoked.
 - See more details on the [state_variables](#) page.

Variable Domains

Continuous variables are typically defined by bounds. Discrete variables can be defined in one of three ways, which are discussed on the page [discrete_variables](#).

Ordering of Variables

The ordering of variables is important, and a consistent ordering is employed throughout the Dakota software. The ordering is shown in `dakota.input.summary` (and in the hierarchical order of this reference manual) and can be summarized as:

1. design
 - (a) continuous
 - (b) discrete integer
 - (c) discrete string
 - (d) discrete real
2. aleatory uncertain
 - (a) continuous
 - (b) discrete integer
 - (c) discrete string
 - (d) discrete real
3. epistemic uncertain
 - (a) continuous
 - (b) discrete integer
 - (c) discrete string
 - (d) discrete real

4. state

- (a) continuous
- (b) discrete integer
- (c) discrete string
- (d) discrete real

Ordering of variable types below this granularity (e.g., from normal to histogram bin within aleatory uncertain - continuous) is defined somewhat arbitrarily, but is enforced consistently throughout the code.

Active Variables

The reason variable types exist is that methods have the capability to treat variable types differently. All methods have default behavior that determines which variable types are "active" and will be assigned values by the method. For example, optimization methods will only vary the design variables - by default.

The default behavior should be described on each method page, or on topics pages that relate to classes of methods. In addition, the default behavior can be modified using the [active](#) keyword.

At least one type of variables that are active for the method in use must have nonzero size (at least 1 active variable) or an input error message will result.

Inferred Default Values and Bounds

The concept of active variables allows any Dakota variable type to be used in any method context. Some methods, e.g., bound-constrained optimization or multi-dimensional or centered parameter studies, require bounds and/or an initial point on the variables, however uncertain variables may not be naturally defined in terms of these characteristics.

Distribution lower and upper bounds are explicit portions of the normal, lognormal, uniform, loguniform, triangular, and beta specifications, whereas they are implicitly defined for others. For example, bounds are naturally defined for histogram bin, histogram point, and interval variables (from the extreme values within the bin/point/interval specifications) as well as for binomial (0 to `num_trials`) and hypergeometric (0 to `min(num_drawn, num_selected)`) variables.

If not specified, distribution bounds are also inferred for normal and lognormal (if optional bounds are unspecified) as well as for exponential, gamma, gumbel, frechet, weibull, poisson, negative binomial, and geometric (which have no bounds specifications); these bounds are $[0, \mu + 3\sigma]$ for exponential, gamma, frechet, weibull, poisson, negative binomial, geometric, and unspecified lognormal, and $[\mu - 3\sigma, \mu + 3\sigma]$ for gumbel and unspecified normal.

When an initial point is needed and not explicitly specified in user input, it is assigned as described in the `initial_point` or `initial_state` specification, e.g., [initial_point](#). For example, uncertain variables are initialized to their means, where mean values for bounded normal and bounded lognormal may be further adjusted to satisfy any user-specified distribution bounds, mean values for discrete integer range distributions are rounded down to the nearest integer, and mean values for discrete set distributions are rounded to the nearest set value.

Examples

Several examples follow. In the first example, two continuous design variables are specified:

```
variables,
  continuous_design = 2
  initial_point      0.9    1.1
  upper_bounds      5.8    2.9
  lower_bounds      0.5   -2.9
  descriptors       'radius' 'location'
```

In the next example, defaults are employed. In this case, `initial_point` will default to a vector of 0. values, `upper_bounds` will default to vector values of `DBL_MAX` (the maximum number representable in double precision for a particular platform), `lower_bounds` will default to a vector of `-DBL_MAX` values, and `descriptors` will default to a vector of `'cdv_i'` strings, where `i` ranges from one to two:

```
variables,
  continuous_design = 2
```

In the following example, the syntax for a normal-lognormal distribution is shown. One normal and one lognormal uncertain variable are completely specified by their means and standard deviations. In addition, the dependence structure between the two variables is specified using the `uncertain_correlation_matrix`.

```
variables,
  normal_uncertain = 1
  means = 1.0
  std_deviations = 1.0
  descriptors = 'TF1n'
  lognormal_uncertain = 1
  means = 2.0
  std_deviations = 0.5
  descriptors = 'TF2ln'
  uncertain_correlation_matrix = 1.0 0.2
                                0.2 1.0
```

An example of the syntax for a state variables specification follows:

```
variables,
  continuous_state = 1
  initial_state = 4.0
  lower_bounds = 0.0
  upper_bounds = 8.0
  descriptors = 'CS1'
  discrete_state_range = 1
  initial_state = 104
  lower_bounds = 100
  upper_bounds = 110
  descriptors = 'DS1'
```

And in a more advanced example, a variables specification containing a set identifier, continuous and discrete design variables, normal and uniform uncertain variables, and continuous and discrete state variables is shown:

```
variables,
  id_variables = 'V1'
  continuous_design = 2
  initial_point = 0.9 1.1
  upper_bounds = 5.8 2.9
  lower_bounds = 0.5 -2.9
  descriptors = 'radius' 'location'
  discrete_design_range = 1
  initial_point = 2
  upper_bounds = 1
  lower_bounds = 3
  descriptors = 'material'
  normal_uncertain = 2
  means = 248.89, 593.33
  std_deviations = 12.4, 29.7
  descriptors = 'TF1n' 'TF2n'
  uniform_uncertain = 2
  lower_bounds = 199.3, 474.63
  upper_bounds = 298.5, 712.
  descriptors = 'TF1u' 'TF2u'
  continuous_state = 2
  initial_state = 1.e-4 1.e-6
  descriptors = 'EPSIT1' 'EPSIT2'
  discrete_state_set
  integer = 1
  initial_state = 100
  set_values = 100 212 375
  descriptors = 'load_case'
```

7.4.1 id_variables

- [Keywords Area](#)

- [variables](#)
- [id_variables](#)

Name the variables block; helpful when there are multiple

Topics

This keyword is related to the topics:

- [block_identifier](#)

Specification

Alias: none

Argument(s): STRING

Default: use of last variables parsed

Description

The optional `id_variables` keyword accepts a string that uniquely identifies this variables block. A model can then use these variables by specifying the same string in its `variables_pointer` specification.

Default Behavior

If the `id_variables` specification is omitted, a particular variables specification will be used by a model only if that model does not include an `variables_pointer` and the variables block was the last (or only) one parsed.

Usage Tips

- It is a best practice to always use explicit variables IDs and pointers to avoid confusion.
- If only one variables block exists, then `id_variables` can be safely omitted from the variables block (and `variables_pointer` omitted from the model specification(s)), since there is no ambiguity.

Examples

For example, a model specification including

```
model
  variables_pointer = 'V1'
```

will link to a response set with

```
  id_variables = 'V1'
```

7.4.2 active

- [Keywords Area](#)
- [variables](#)
- [active](#)

Set the active variables view a method will see

Specification

Alias: none

Argument(s): none

Default: Infer from response or method specification

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
			all	Option for the active keyword
	Required (<i>Choose One</i>)	Active Variables (Group 1)	design	Option for the active keyword
			uncertain	Option for the active keyword
			aleatory	Option for the active keyword
			epistemic	Option for the active keyword
			state	Option for the active keyword

Description

There are certain situations where the user may want to explicitly control the subset of variables that is considered active for a certain Dakota method. This is done by specifying the keyword `active` in the variables specification block, followed by one of the following: `all`, `design`, `uncertain`, `aleatory`, `epistemic`, or `state`.

Specifying one of these subsets of variables will allow the Dakota method to operate on the specified variable types and override the default active subset.

If the user does not specify any explicit override of the active view of the variables, Dakota first considers the response function specification.

- If the user specifies objective functions or calibration terms in the response specification block, then we can infer that the active variables should be the design variables (since design variables are used within optimization and least squares methods).
- If the user instead specifies the generic response type of `response_functions`, then Dakota cannot infer the active variable subset from the response specification and will instead infer it from the method selection.
 1. If the method is a parameter study, or any of the methods available under `dace`, `psuade`, or `fsu` methods, the active view is set to all variables.
 2. For uncertainty quantification methods, if the method is sampling, then the view is set to `aleatory` if only `aleatory` variables are present, `epistemic` if only `epistemic` variables are present, or `uncertain` (covering both `aleatory` and `epistemic`) if both are present.
 3. If the uncertainty method involves interval estimation or evidence calculations, the view is set to `epistemic`.
 4. For other uncertainty quantification methods not mentioned in the previous sentences (e.g., reliability methods or stochastic expansion methods), the default view is set to `aleatory`.
 5. Finally, for verification studies using the Richardson extrapolation method, the active view is set to `state`.
 6. Note that in surrogate-based optimization, where the surrogate is built on points defined by the method defined by the `dace_method_pointer`, the sampling used to generate the points is performed only over the design variables as a default unless otherwise specified (e.g. `state` variables will not be sampled for surrogate construction).

As alluded to in the previous section, the iterative method selected for use in Dakota determines what subset, or view, of the variables data is active in the iteration. The general case of having a mixture of various different types of variables is supported within all of the Dakota methods even though certain methods will only modify certain types of variables (e.g., optimizers and least squares methods only modify design variables, and uncertainty quantification methods typically only utilize uncertain variables). This implies that variables which are not under the direct control of a particular iterator will be mapped through the interface in an unmodified state. This allows for a variety of parameterizations within the model in addition to those which are being used by a particular iterator, which can provide the convenience of consolidating the control over various modeling parameters in a single file (the Dakota input file). An important related point is that the variable set that is active with a particular iterator is the same variable set for which derivatives are typically computed.

Examples

For example, the default behavior for a nondeterministic sampling method is to sample the uncertain variables. However, if the user specified `active all` in the variables specification block, the sampling would be performed over all variables (e.g. design and state variables in addition to the uncertain variables). This may be desired in situations such as surrogate-based optimization under uncertainty, where a surrogate may be constructed to span both design and uncertain variables. This is an example where we expand the active subset beyond the default, but in other situations, we may wish to restrict from the default. An example of this would be performing design of experiments in the presence of multiple variable types (for which all types are active by default), but only wanting to sample over the design variables for purposes of constructing a surrogate model for optimization.

Theory

The optional status of the different variable type specifications allows the user to specify only those variables which are present (rather than explicitly specifying that the number of a particular type of variables is zero). However, at least one type of variables that are active for the iterator in use must have nonzero size or an input error message will result.

7.4.2.1 all

- [Keywords Area](#)
- [variables](#)
- [active](#)
- [all](#)

Option for the `active` keyword

Specification

Alias: none

Argument(s): none

Description

See the [active](#) keyword

7.4.2.2 design

- [Keywords Area](#)

- [variables](#)
- [active](#)
- [design](#)

Option for the `active` keyword

Specification

Alias: none

Argument(s): none

Description

See the [active](#) keyword

7.4.2.3 uncertain

- [Keywords Area](#)
- [variables](#)
- [active](#)
- [uncertain](#)

Option for the `active` keyword

Specification

Alias: none

Argument(s): none

Description

See the [active](#) keyword

7.4.2.4 aleatory

- [Keywords Area](#)
- [variables](#)
- [active](#)
- [aleatory](#)

Option for the `active` keyword

Specification

Alias: none

Argument(s): none

Description

See the [active](#) keyword

7.4.2.5 epistemic

- [Keywords Area](#)
- [variables](#)
- [active](#)
- [epistemic](#)

Option for the `active` keyword

Specification

Alias: none

Argument(s): none

Description

See the [active](#) keyword

7.4.2.6 state

- [Keywords Area](#)
- [variables](#)
- [active](#)
- [state](#)

Option for the `active` keyword

Specification

Alias: none

Argument(s): none

Description

See the [active](#) keyword

7.4.3 mixed

- [Keywords Area](#)
- [variables](#)
- [mixed](#)

Maintain continuous/discrete variable distinction

Specification

Alias: none

Argument(s): none

Default: relaxed (branch and bound), mixed (all other methods)

Description

The variables domain specifies how the discrete variables are treated. If the user specifies `mixed` in the variable specification block, the continuous and discrete variables are treated separately. If the user specifies `relaxed` in the variable specification block, the discrete variables are relaxed and treated as continuous variables. This may be useful in optimization problems involving both continuous and discrete variables when a user would like to use an optimization method that is designed for continuous variable optimization. All Dakota methods have a default value of `mixed` for the domain type except for the branch-and-bound method which has a default domain type of `relaxed`. Note that the branch-and-bound method is under development at this time. Finally, note that the domain selection applies to all variable types: design, aleatory uncertain, epistemic uncertain, and state.

With respect to domain type, if the user does not specify an explicit override of `mixed` or `relaxed`, Dakota infers the domain type from the method. As mentioned above, all methods currently use a `mixed` domain as a default, except the branch-and-bound method which is under development.

See Also

These keywords may also be of interest:

- [relaxed](#)

7.4.4 relaxed

- [Keywords Area](#)
- [variables](#)
- [relaxed](#)

Allow treatment of discrete variables as continuous

Specification

Alias: none

Argument(s): none

Description

The variables domain specifies how the discrete variables are treated. If the user specifies `mixed` in the variable specification block, the continuous and discrete variables are treated separately. If the user specifies `relaxed` in the variable specification block, the discrete variables are relaxed and treated as continuous variables. This may be useful in optimization problems involving both continuous and discrete variables when a user would like to use an optimization method that is designed for continuous variable optimization. All Dakota methods have a default value of `mixed` for the domain type except for the branch-and-bound method which has a default domain type of `relaxed`. Note that the branch-and-bound method is under development at this time. Finally, note that the domain selection applies to all variable types: design, aleatory uncertain, epistemic uncertain, and state.

With respect to domain type, if the user does not specify an explicit override of `mixed` or `relaxed`, Dakota infers the domain type from the method. As mentioned above, all methods currently use a mixed domain as a default, except the branch-and-bound method which is under development.

See Also

These keywords may also be of interest:

- [mixed](#)

7.4.5 continuous_design

- [Keywords Area](#)
- [variables](#)
- [continuous_design](#)

Design variable - continuous

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [design_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no continuous design variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_point	Initial values for variables
	Optional		lower_bounds	Specify minimum values
	Optional		upper_bounds	Specify maximum values
	Optional		scale_types	How to scale each continuous design variable

	Optional	scales	Characteristic values to scale continuous design variables
	Optional	descriptors	Labels for the variables

Description

Continuous variables are defined by a real interval and are changed during the search for the optimal design.

7.4.5.1 initial_point

- [Keywords Area](#)
- [variables](#)
- [continuous_design](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: `cdv_initial_point`

Argument(s): REALLIST

Default: 0.0

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.5.2 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [continuous_design](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: cdv_lower_bounds

Argument(s): REALLIST

Default: -infinity

Description

Specify minimum values

7.4.5.3 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [continuous_design](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: cdv_upper_bounds

Argument(s): REALLIST

Default: infinity

Description

Specify maximum values

7.4.5.4 scale_types

- [Keywords Area](#)
- [variables](#)
- [continuous_design](#)
- [scale_types](#)

How to scale each continuous design variable

Specification

Alias: `cdv_scale_types`

Argument(s): STRINGLIST

Default: vector values = 'none'

Description

Each string in `scale_types` indicates the scaling type for each continuous design variable. They only have effect when the associated method specifies `scaling`.

The options are:

- 'value' - characteristic value by which variables will be divided. If this is chosen, then [scales](#) must also be specified; 'value' is assumed if scales are given without `scale_types`
- 'auto' - automatic scaling based on bounds.
- 'log' - logarithmic scaling (can be used together with [scales](#)).

If a single string is specified it will apply to all continuous design variables. Otherwise, a scale type must be specified for each continuous design variable.

Usage Tips:

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

Examples

Two continuous design variables, one scaled by 4.0, the other by 0.1, then further log scaled:

```
continuous_design = 2
  initial_point   -1.2      1.0
  lower_bounds    -2.0      0.001
  upper_bounds    2.0       2.0
  descriptors     'x1'      "x2"
  scale_types =   'value'  'log'
  scales =        4.0      0.1
```

7.4.5.5 scales

- [Keywords Area](#)
- [variables](#)
- [continuous_design](#)
- [scales](#)

Characteristic values to scale continuous design variables

Specification

Alias: `cdv_scales`

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

Description

Each real value in `scales` is a nonzero characteristic value to be used in scaling each continuous design variable. They only have effect when the associated method specifies `scaling`.

This keyword is required for `scale_types` of `'value'` and optional for `'auto'` and `'log'`. When specified in conjunction with `log`, scale values are applied prior to the logarithm, to permit log scaling of strictly negative design variables.

If a single real value is specified it will apply to all continuous design variables. Otherwise, a scale value must be specified for each continuous design variable.

Usage Tips:

When `scales` are specified, but not `scale_types`, the scaling type is assumed to be `'value'` for all continuous design variables.

Use scale values of 1.0 to selectively avoid scaling a subset of the variables.

Scaling for linear constraints is applied *after* any continuous variable scaling.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

Examples

Two continuous design variables, one scaled by the characteristic value 4.0, the other log-scaled without additional value scaling

```
continuous_design = 2
initial_point     -1.2      1.0
lower_bounds     -200      0.001
upper_bounds     200       2.0
descriptors      'x1'      "x2"
scale_types      'value' 'log'
scales           = 4.0 1.0
```

7.4.5.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [continuous_design](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: `cdv_descriptors`

Argument(s): STRINGLIST

Default: `cdv_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.6 discrete_design_range

- [Keywords Area](#)
- [variables](#)
- [discrete_design_range](#)

Design variable - discrete range-valued

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [design_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete design variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_point	Initial values for variables
	Optional		lower_bounds	Specify minimum values
	Optional		upper_bounds	Specify maximum values
	Optional		descriptors	Labels for the variables

Description

These variables take on a range of integer values from the specified lower bound to the specified upper bound (integer interval). The details of how to specify this discrete variable are located on the [discrete_variables](#) page.

7.4.6.1 initial_point

- [Keywords Area](#)
- [variables](#)
- [discrete_design_range](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: `ddv_initial_point`

Argument(s): INTEGERLIST

Default: 0

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.6.2 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [discrete_design_range](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: `ddv_lower_bounds`

Argument(s): INTEGERLIST

Default: INT_MIN

Description

Specify minimum values

7.4.6.3 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [discrete_design_range](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: `ddv_upper_bounds`

Argument(s): INTEGERLIST

Default: `INT_MAX`

Description

Specify maximum values

7.4.6.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_design_range](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: `ddv_descriptors`

Argument(s): STRINGLIST

Default: `ddriv_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.7 discrete_design_set

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)

Design variable - discrete set-valued

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [design_variables](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		integer	Integer-valued discrete design variables
	Optional		string	String-valued discrete design set variables
	Optional		real	Real-valued discrete design variables

Description

Discrete design variables whose values come from a set of admissible elements. Each variable specified must be of type integer, string, or real.

7.4.7.1 [integer](#)

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [integer](#)

Integer-valued discrete design variables

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [design_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete design set integer variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_variable	Number of admissible elements for each set variable
	Required		elements	The permissible values for each discrete variable
	Optional		categorical	Whether the set-valued variables are categorical or relaxable
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

A design variable whose values come from a specified set of admissible integers. The details of how to specify this discrete variable are located on the [discrete_variables](#) page.

Examples

Four integer variables whose values will be selected from the following sets during the search for an optimal design. $y_1 \in \{0, 1\}$, $y_2 \in \{0, 1\}$, $y_3 \in \{0, 5\}$ and $y_4 \in \{10, 15, 20, 23\}$.

```
discrete_design_set
integer 4
  descriptors      'y1'  'y2'  'y3'  'y4'
  elements_per_variable 2    2    2    4
  elements         0 1   0 1   0 5   10 15 20 23
```

7.4.7.1.1 elements_per_variable

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [integer](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.7.1.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [integer](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): INTEGERLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.7.1.3 categorical

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [integer](#)
- [categorical](#)

Whether the set-valued variables are categorical or relaxable

Specification**Alias:** none**Argument(s):** STRINGLIST**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		adjacency_matrix	1-0 matrix defining which categorical variable levels are related.

Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```
discrete_design_set
  integer 1
    elements 2 4 7
  descriptor 'rotor_blades'
  categorical 'no'
```

7.4.7.1.3.1 adjacency_matrix

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [integer](#)
- [categorical](#)
- [adjacency_matrix](#)

1-0 matrix defining which categorical variable levels are related.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The `adjacency_matrix` keyword is associated with `discrete_design_set` variables that are specified to be categorical. Each such variable is associated with one $k \times k$ symmetric matrix, where k is the number of values (or levels) of the variable. Entry i, j of a matrix should be 1 if level i and level j are related by some subjective criteria or if $i = j$; it should be 0 otherwise. The matrices for all variables of the same type (string, real, or integer) are entered sequentially as a list of integers as shown in the examples below.

Default Behavior

The `adjacency_matrix` keyword is only relevant for `discrete_design_set` real and `discrete_design_set` integer variables if one or more of them have been specified to be categorical. It is always relevant for `discrete_design_set` string variables. If the user does not define an adjacency matrix, the default is method dependent. Currently, the only method that makes use of the adjacency matrix is [mesh_adaptive_search](#), which uses a tri-diagonal adjacency matrix by default.

Expected Output

The expected output is method dependent.

Usage Tips

If an adjacency matrix is defined for one type of (categorical) `discrete_design_set` variable, it must be defined for all variables of that type, even for those not defined to be categorical. Those for the non-categorical set variables will be ignored.

Examples

The following example shows a variables specification where some real and some integer `discrete_design_set` variables are categorical.

```
variables
  continuous_design = 3
    initial_point -1.0 1.5 2.0
    lower_bounds -10.0 -10.0 -10.0
    upper_bounds 10.0 10.0 10.0
    descriptors 'x1' 'x2' 'x3'
  discrete_design_range = 2
    initial_point 2 2
    lower_bounds 1 1
    upper_bounds 4 9
    descriptors 'y1' 'y2'
  discrete_design_set
    real = 2
      elements_per_variable = 4 5
      elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
      descriptors 'y3' 'y4'
      categorical 'no' 'yes'
      adjacency_matrix 1 1 0 0 # Begin entry of 4x4 matrix for y3
                      1 1 1 0
                      0 1 1 1
                      0 0 1 1
                      1 0 1 0 1 # Begin entry of 5x5 matrix for y4
                      0 1 0 1 0
                      1 0 1 0 1
                      0 1 0 1 0
                      1 0 1 0 1
    integer = 2
      elements_per_variable = 2 3
      elements = 4 7 8 9 12
      descriptors 'z1' 'z2'
      categorical 'yes' 'yes'
```

Note that for the real case, the user wants to define an adjacency matrix for the categorical variable, so adjacency matrices for both variables must be specified. The matrix for the first one will be ignored. Note that no adjacency matrix is specified for either integer categorical variable. The default will be used in both cases. Currently the only method taking advantage of adjacency matrices is `mesh_adaptive_search`, which uses a tri-diagonal adjacency matrix by default. Thus, the matrices used would be

```
z1: 1 1
    1 1
z2: 1 1 0
    1 1 1
    0 1 1
```

The following example shows a variables specification for string variables. Note that string variables are always

considered to be categorical. If an adjacency matrix is not specified, a method-dependent default matrix will be used.

```
variables,
  continuous_design = 2
  initial_point 0.5 0.5
  lower_bounds 0. 0.
  upper_bounds 1. 1.
  descriptors = 'x' 'y'
discrete_design_set string = 1
  elements = 'aniso1' 'aniso2' 'iso1' 'iso2' 'iso3'
  descriptors = 'ancomp'
  adjacency_matrix 1 1 0 0 0
                  1 1 0 0 0
                  0 0 1 1 1
                  0 0 1 1 1
                  0 0 1 1 1
```

See Also

These keywords may also be of interest:

- [mesh_adaptive_search](#)

7.4.7.1.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [integer](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Default: middle set value, or rounded down

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.7.1.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [integer](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.7.2 string

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [string](#)

String-valued discrete design set variables

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [design_variables](#)

Specification**Alias:** none**Argument(s):** INTEGER**Default:** no discrete design set string variables**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_variable	Number of admissible elements for each set variable
	Required		elements	The permissible values for each discrete variable
	Optional		adjacency_matrix	1-0 matrix defining which categorical variable levels are related.
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

Discrete design variables whose values come from a specified set of admissible strings. The details of how to specify this discrete variable are located on the [discrete_variables](#) page. Each string element value must be quoted and may contain alphanumeric, dash, underscore, and colon. White space, quote characters, and backslash/metacharacters are not permitted.

Examples

Two string variables whose values will be selected from the set of provided elements. The first variable, 'linear solver', takes on values from a set of three possible elements and the second variable, 'mesh_file', from a set of two possible elements.

```
discrete_design_set
string 2
  descriptors      'linear_solver'  'mesh_file'
  elements_per_variable 3          2
  elements         'cg' 'gmres' 'direct'
                  'mesh64.exo' 'mesh128.exo'
```

7.4.7.2.1 elements_per_variable

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [string](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.7.2.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [string](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): STRINGLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.7.2.3 adjacency_matrix

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [string](#)
- [adjacency_matrix](#)

1-0 matrix defining which categorical variable levels are related.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The `adjacency_matrix` keyword is associated with `discrete_design_set` variables that are specified to be categorical. Each such variable is associated with one $k \times k$ symmetric matrix, where k is the number of values (or levels) of the variable. Entry i, j of a matrix should be 1 if level i and level j are related by some subjective criteria or if $i = j$; it should be 0 otherwise. The matrices for all variables of the same type (string, real, or integer) are entered sequentially as a list of integers as shown in the examples below.

Default Behavior

The `adjacency_matrix` keyword is only relevant for `discrete_design_set` real and `discrete_design_set` integer variables if one or more of them have been specified to be categorical. It is always relevant for `discrete_design_set` string variables. If the user does not define an adjacency matrix, the default is method dependent. Currently, the only method that makes use of the adjacency matrix is [mesh_adaptive_search](#), which uses a tri-diagonal adjacency matrix by default.

Expected Output

The expected output is method dependent.

Usage Tips

If an adjacency matrix is defined for one type of (categorical) `discrete_design_set` variable, it must be defined for all variables of that type, even for those not defined to be categorical. Those for the non-categorical set variables will be ignored.

Examples

The following example shows a variables specification where some real and some integer `discrete_design_set` variables are categorical.

```
variables
  continuous_design = 3
  initial_point -1.0 1.5 2.0
  lower_bounds -10.0 -10.0 -10.0
  upper_bounds 10.0 10.0 10.0
  descriptors 'x1' 'x2' 'x3'
  discrete_design_range = 2
  initial_point 2 2
  lower_bounds 1 1
  upper_bounds 4 9
  descriptors 'y1' 'y2'
  discrete_design_set
  real = 2
  elements_per_variable = 4 5
  elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
  descriptors 'y3' 'y4'
  categorical 'no' 'yes'
  adjacency_matrix 1 1 0 0 # Begin entry of 4x4 matrix for y3
                  1 1 1 0
                  0 1 1 1
                  0 0 1 1
                  1 0 1 0 1 # Begin entry of 5x5 matrix for y4
                  0 1 0 1 0
                  1 0 1 0 1
                  0 1 0 1 0
                  1 0 1 0 1
  integer = 2
  elements_per_variable = 2 3
  elements = 4 7 8 9 12
  descriptors 'z1' 'z2'
  categorical 'yes' 'yes'
```

Note that for the real case, the user wants to define an adjacency matrix for the categorical variable, so adjacency matrices for both variables must be specified. The matrix for the first one will be ignored. Note that no adjacency matrix is specified for either integer categorical variable. The default will be used in both cases. Currently the

only method taking advantage of adjacency matrices is `mesh_adaptive_search`, which uses a tri-diagonal adjacency matrix by default. Thus, the matrices used would be

```
z1: 1 1
    1 1
z2: 1 1 0
    1 1 1
    0 1 1
```

The following example shows a variables specification for string variables. Note that string variables are always considered to be categorical. If an adjacency matrix is not specified, a method-dependent default matrix will be used.

```
variables,
  continuous_design = 2
  initial_point 0.5 0.5
  lower_bounds 0. 0.
  upper_bounds 1. 1.
  descriptors = 'x' 'y'
  discrete_design_set string = 1
  elements = 'aniso1' 'aniso2' 'iso1' 'iso2' 'iso3'
  descriptors = 'ancomp'
  adjacency_matrix 1 1 0 0 0
                  1 1 0 0 0
                  0 0 1 1 1
                  0 0 1 1 1
                  0 0 1 1 1
```

See Also

These keywords may also be of interest:

- [mesh_adaptive_search](#)

7.4.7.2.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [string](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: middle set value, or rounded down

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.7.2.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [string](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.7.3 real

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [real](#)

Real-valued discrete design variables

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [design_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete design set real variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_variable	Number of admissible elements for each set variable
	Required		elements	The permissible values for each discrete variable
	Optional		categorical	Whether the set-valued variables are categorical or relaxable
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

A design variable whose values come from a specified set of admissible reals. The details of how to specify this discrete variable are located on the [discrete_variables](#) page.

Examples

Two continuous, restricted variables whose values will be selected from the following sets during the search for an optimal design. $y_1 \in \{0.25, 1.25, 2.25, 3.25, 4.25\}$, $y_2 \in \{0, 5\}$

```
discrete_design_set
  real 2
    descriptors          'y1'                'y2'
    elements_per_variable 5                    2
    elements             0.25 1.25 2.25 3.25 4.25 0 5
```

7.4.7.3.1 elements_per_variable

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [real](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.7.3.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [real](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): REALLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.7.3.3 categorical

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [real](#)
- [categorical](#)

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none

Argument(s): STRINGLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		adjacency_matrix	1-0 matrix defining which categorical variable levels are related.

Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```
discrete_design_set
  integer 1
  elements 2 4 7
  descriptor 'rotor_blades'
  categorical 'no'
```

7.4.7.3.3.1 adjacency_matrix

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [real](#)
- [categorical](#)
- [adjacency_matrix](#)

1-0 matrix defining which categorical variable levels are related.

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The `adjacency_matrix` keyword is associated with `discrete_design_set` variables that are specified to be categorical. Each such variable is associated with one $k \times k$ symmetric matrix, where k is the number of values (or levels) of the variable. Entry i, j of a matrix should be 1 if level i and level j are related by some subjective criteria or if $i = j$; it should be 0 otherwise. The matrices for all variables of the same type (string, real, or integer) are entered sequentially as a list of integers as shown in the examples below.

Default Behavior

The `adjacency_matrix` keyword is only relevant for `discrete_design_set` real and `discrete_design_set integer` variables if one or more of them have been specified to be categorical. It is always relevant for `discrete_design_set string` variables. If the user does not define an adjacency matrix, the default is method dependent. Currently, the only method that makes use of the adjacency matrix is [mesh_adaptive_search](#), which uses a tri-diagonal adjacency matrix by default.

Expected Output

The expected output is method dependent.

Usage Tips

If an adjacency matrix is defined for one type of (categorical) `discrete_design_set` variable, it must be defined for all variables of that type, even for those not defined to be categorical. Those for the non-categorical set variables will be ignored.

Examples

The following example shows a variables specification where some real and some integer `discrete_design_set` variables are categorical.

```
variables
  continuous_design = 3
  initial_point -1.0 1.5 2.0
  lower_bounds -10.0 -10.0 -10.0
  upper_bounds 10.0 10.0 10.0
  descriptors 'x1' 'x2' 'x3'
  discrete_design_range = 2
  initial_point 2 2
  lower_bounds 1 1
  upper_bounds 4 9
  descriptors 'y1' 'y2'
  discrete_design_set
  real = 2
  elements_per_variable = 4 5
  elements = 1.2 2.3 3.4 4.5 1.2 3.3 4.4 5.5 7.7
  descriptors 'y3' 'y4'
  categorical 'no' 'yes'
  adjacency_matrix 1 1 0 0 # Begin entry of 4x4 matrix for y3
                  1 1 1 0
                  0 1 1 1
                  0 0 1 1
                  1 0 1 0 1 # Begin entry of 5x5 matrix for y4
                  0 1 0 1 0
                  1 0 1 0 1
                  0 1 0 1 0
                  1 0 1 0 1
  integer = 2
  elements_per_variable = 2 3
  elements = 4 7 8 9 12
```

```

descriptors 'z1' 'z2'
categorical 'yes' 'yes'

```

Note that for the real case, the user wants to define an adjacency matrix for the categorical variable, so adjacency matrices for both variables must be specified. The matrix for the first one will be ignored. Note that no adjacency matrix is specified for either integer categorical variable. The default will be used in both cases. Currently the only method taking advantage of adjacency matrices is `mesh_adaptive_search`, which uses a tri-diagonal adjacency matrix by default. Thus, the matrices used would be

```

z1: 1 1
    1 1
z2: 1 1 0
    1 1 1
    0 1 1

```

The following example shows a variables specification for string variables. Note that string variables are always considered to be categorical. If an adjacency matrix is not specified, a method-dependent default matrix will be used.

```

variables,
  continuous_design = 2
  initial_point 0.5 0.5
  lower_bounds 0. 0.
  upper_bounds 1. 1.
  descriptors = 'x' 'y'
  discrete_design_set string = 1
  elements = 'aniso1' 'aniso2' 'iso1' 'iso2' 'iso3'
  descriptors = 'ancomp'
  adjacency_matrix 1 1 0 0 0
                  1 1 0 0 0
                  0 0 1 1 1
                  0 0 1 1 1
                  0 0 1 1 1

```

See Also

These keywords may also be of interest:

- [mesh_adaptive_search](#)

7.4.7.3.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [real](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Default: middle set value, or rounded down

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.7.3.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_design_set](#)
- [real](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.8 normal_uncertain

- [Keywords Area](#)
- [variables](#)
- [normal_uncertain](#)

Aleatory uncertain variable - normal (Gaussian)

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no normal uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		means	First parameter of the distribution
	Required		std_deviations	Second parameter of the distribution
	Optional		lower_bounds	Specify minimum values
	Optional		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The number of normal uncertain variables, their means, and standard deviations are required specifications, while the distribution lower and upper bounds and variable descriptors are optional specifications. The normal distribution is widely used to model uncertain variables such as population characteristics. It is also used to model the mean of a sample: as the sample size becomes very large, the Central Limit Theorem states that the distribution of the mean becomes approximately normal, regardless of the distribution of the original variables.

The density function for the normal distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

where μ and σ are the mean and standard deviation of the normal distribution, respectively.

Note that if you specify bounds for a normal distribution, the sampling occurs from the underlying distribution with the given mean and standard deviation, but samples are not taken outside the bounds (see "bounded normal" distribution type in [93]). This can result in the mean and the standard deviation of the sample data being different from the mean and standard deviation of the underlying distribution. For example, if you are sampling from a normal distribution with a mean of 5 and a standard deviation of 3, but you specify bounds of 1 and 7, the resulting mean of the samples will be around 4.3 and the resulting standard deviation will be around 1.6. This is because you have bounded the original distribution significantly, and asymmetrically, since 7 is closer to the original mean than 1.

Theory

When used with some methods such as design of experiments and multidimensional parameter studies, distribution bounds are inferred to be $[\mu - 3\sigma, \mu + 3\sigma]$

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means, correcting to bounds if needed.

7.4.8.1 means

- [Keywords Area](#)
- [variables](#)
- [normal_uncertain](#)
- [means](#)

First parameter of the distribution

Specification

Alias: nuv_means

Argument(s): REALLIST

Description

Means

7.4.8.2 std_deviations

- [Keywords Area](#)
- [variables](#)
- [normal_uncertain](#)
- [std_deviations](#)

Second parameter of the distribution

Specification

Alias: nuv_std_deviations

Argument(s): REALLIST

Description

Standard deviation

7.4.8.3 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [normal_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: nuv_lower_bounds

Argument(s): REALLIST

Default: -infinity

Description

Specify minimum values

7.4.8.4 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [normal_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: nuv_upper_bounds

Argument(s): REALLIST

Default: infinity

Description

Specify maximum values

7.4.8.5 initial_point

- [Keywords Area](#)
- [variables](#)
- [normal_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.8.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [normal_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: nuv_descriptors

Argument(s): STRINGLIST

Default: nuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.9 lognormal_uncertain

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)

Aleatory uncertain variable - lognormal

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no lognormal uncertain variables

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Lognormal Characterization (Group 1)	Dakota Keyword lambdas	Dakota Keyword Description First parameter of the lognormal distribution (option 3)
			means	First parameter of the lognormal distribution (options 1 & 2)
	Optional		lower_bounds	Specify minimum values
	Optional		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

If the logarithm of an uncertain variable X has a normal distribution, that is $\log X \sim N(\mu, \sigma)$, then X is distributed with a lognormal distribution. The lognormal is often used to model:

1. time to perform some task
2. variables which are the product of a large number of other quantities, by the Central Limit Theorem
3. quantities which cannot have negative values.

The number of lognormal uncertain variables, their means, and either standard deviations or error factors must be specified, while the distribution lower and upper bounds and variable descriptors are optional specifications. These distribution bounds can be used to truncate the tails of lognormal distributions, which as for bounded normal, can result in the mean and the standard deviation of the sample data being different from the mean and standard deviation of the underlying distribution (see "bounded lognormal" and "bounded lognormal-n" distribution types in [93]).

For the lognormal variables, one may specify either the mean μ and standard deviation σ of the actual lognormal distribution (option 1), the mean μ and error factor ε of the actual lognormal distribution (option 2), or the mean λ ("lambda") and standard deviation ζ ("zeta") of the underlying normal distribution (option 3).

The conversion equations from lognormal mean μ and either lognormal error factor ε or lognormal standard deviation σ to the mean λ and standard deviation ζ of the underlying normal distribution are as follows:

$$\zeta = \frac{\ln(\varepsilon)}{1.645}$$

$$\zeta^2 = \ln\left(\frac{\sigma^2}{\mu^2} + 1\right)$$

$$\lambda = \ln(\mu) - \frac{\zeta^2}{2}$$

Conversions from λ and ζ back to μ and ε or σ are as follows:

$$\mu = \exp\left(\lambda + \frac{\zeta^2}{2}\right)$$

$$\sigma^2 = \exp(2\lambda + \zeta^2) (\exp(\zeta^2) - 1)$$

$$\varepsilon = \exp(1.645\zeta)$$

The density function for the lognormal distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi}\zeta x} \exp\left(-\frac{1}{2}\left(\frac{\ln(x) - \lambda}{\zeta}\right)^2\right)$$

Theory

When used with some methods such as design of experiments and multidimensional parameter studies, distribution bounds are inferred to be $[0, \mu + 3\sigma]$.

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means, correcting to bounds if needed.

7.4.9.1 lambdas

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [lambdas](#)

First parameter of the lognormal distribution (option 3)

Specification**Alias:** Inuv_lambdas**Argument(s):** REALLIST**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		zetas	Second parameter of the lognormal distribution (option 3)

Description

For the lognormal variables, one may specify the mean λ ("lambda") and standard deviation ζ ("zeta") of the underlying normal distribution.

7.4.9.1.1 zetas

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [lambdas](#)
- [zetas](#)

Second parameter of the lognormal distribution (option 3)

Specification

Alias: Inuv_zetas

Argument(s): REALLIST

Description

For the lognormal variables, one may specify the mean λ ("lambda") and standard deviation ζ ("zeta") of the underlying normal distribution.

7.4.9.2 means

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [means](#)

First parameter of the lognormal distribution (options 1 & 2)

Specification

Alias: Inuv_means

Argument(s): REALLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Spread (Group 1)	std_deviations	Second parameter of the lognormal distribution (option 1)
			error_factors	Second parameter of the lognormal distribution (option 2)

Description

For the lognormal variables, one may specify either the mean μ and standard deviation σ of the actual lognormal distribution, the mean μ and error factor ε of the actual lognormal distribution.

This corresponds to the mean of the lognormal random variable

7.4.9.2.1 [std_deviations](#)

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [means](#)
- [std_deviations](#)

Second parameter of the lognormal distribution (option 1)

Specification

Alias: `Inuv_std_deviations`

Argument(s): REALLIST

Description

For the lognormal variables, one may specify either the mean μ and standard deviation σ of the actual lognormal distribution.

This corresponds to the standard deviation of the lognormal random variable.

7.4.9.2.2 [error_factors](#)

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [means](#)
- [error_factors](#)

Second parameter of the lognormal distribution (option 2)

Specification

Alias: Inuv_error_factors

Argument(s): REALLIST

Description

For the lognormal variables, one may specify the mean μ and error factor ε of the actual lognormal distribution. This specifies the error function of the lognormal random variable.

7.4.9.3 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: Inuv_lower_bounds

Argument(s): REALLIST

Default: 0

Description

Specify minimum values

7.4.9.4 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: Inuv_upper_bounds

Argument(s): REALLIST

Default: infinity

Description

Specify maximum values

7.4.9.5 initial_point

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.9.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [lognormal_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: Inuv_descriptors

Argument(s): STRINGLIST

Default: Inuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.10 uniform_uncertain

- [Keywords Area](#)
- [variables](#)
- [uniform_uncertain](#)

Aleatory uncertain variable - uniform

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no uniform uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		lower_bounds	Specify minimum values
	Required		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The number of uniform uncertain variables and their distribution lower and upper bounds are required specifications, while variable descriptors is an optional specification. The uniform distribution has the density function:

$$f(x) = \frac{1}{U-L}$$

where U and L are the upper and lower bounds of the uniform distribution, respectively. The mean of the uniform distribution is $\frac{U+L}{2}$ and the variance is $\frac{(U-L)^2}{12}$.

Theory

This distribution is a special case of the more general beta distribution.

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means.

7.4.10.1 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [uniform_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: uuv_lower_bounds

Argument(s): REALLIST

Description

Specify minimum values

7.4.10.2 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [uniform_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: uuv_upper_bounds

Argument(s): REALLIST

Description

Specify maximum values

7.4.10.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [uniform_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.10.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [uniform_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: uuv_descriptors

Argument(s): STRINGLIST

Default: uuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.11 loguniform_uncertain

- [Keywords Area](#)
- [variables](#)
- [loguniform_uncertain](#)

Aleatory uncertain variable - loguniform

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no loguniform uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		lower_bounds	Specify minimum values
	Required		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

If the logarithm of an uncertain variable X has a uniform distribution, that is $\log X \sim U(L_{LU}, U_{LU})$, then X is distributed with a loguniform distribution. The distribution lower bound is L_{LU} and upper bound is U_{LU} . The loguniform distribution has the density function:

$$f(x) = \frac{1}{x(\ln(U_{LU}) - \ln(L_{LU}))}$$

Theory

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means.

7.4.11.1 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [loguniform_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: luuv_lower_bounds

Argument(s): REALLIST

Description

Specify minimum values

7.4.11.2 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [loguniform_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: luuv_upper_bounds

Argument(s): REALLIST

Description

Specify maximum values

7.4.11.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [loguniform_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.11.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [loguniform_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: luuv_descriptors

Argument(s): STRINGLIST

Default: luuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.12 triangular_uncertain

- [Keywords Area](#)
- [variables](#)
- [triangular_uncertain](#)

Aleatory uncertain variable - triangular

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no triangular uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		modes	Distribution parameter
	Required		lower_bounds	Specify minimum values
	Required		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The triangular distribution is often used when one does not have much data or information, but does have an estimate of the most likely value and the lower and upper bounds. The number of triangular uncertain variables, the modes, and the distribution lower and upper bounds are required specifications, while variable descriptors is an optional specification.

The density function for the triangular distribution is:

$$f(x) = \frac{2(x-L)}{(U-L)(M-L)}$$

if $L \leq x \leq M$, and

$$f(x) = \frac{2(U-x)}{(U-L)(U-M)}$$

if $M \leq x \leq U$, and 0 elsewhere. In these equations, L is the lower bound, U is the upper bound, and M is the mode of the triangular distribution.

7.4.12.1 modes

- [Keywords Area](#)
- [variables](#)
- [triangular_uncertain](#)
- [modes](#)

Distribution parameter

Specification

Alias: tuv_modes

Argument(s): REALLIST

Description

Specify the modes

7.4.12.2 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [triangular_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: tuv_lower_bounds

Argument(s): REALLIST

Description

Specify minimum values

7.4.12.3 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [triangular_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: tuv_upper_bounds

Argument(s): REALLIST

Description

Specify maximum values

7.4.12.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [triangular_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.12.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [triangular_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: tuv_descriptors

Argument(s): STRINGLIST

Default: tuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.13 exponential_uncertain

- [Keywords Area](#)
- [variables](#)
- [exponential_uncertain](#)

Aleatory uncertain variable - exponential

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification**Alias:** none**Argument(s):** INTEGER**Default:** no exponential uncertain variables**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		betas	Parameter of the exponential distribution
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The exponential distribution is often used for modeling failure rates.

The density function for the exponential distribution is given by:

$$f(x) = \frac{1}{\beta} \exp\left(\frac{-x}{\beta}\right)$$

where $\mu_E = \beta$ and $\sigma_E^2 = \beta^2$.

Note that this distribution is a special case of the more general gamma distribution.

Theory

When used with some methods such as design of experiments and multidimensional parameter studies, distribution bounds are inferred to be $[0, \mu + 3\sigma]$.

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means

7.4.13.1 betas

- [Keywords Area](#)
- [variables](#)
- [exponential_uncertain](#)
- [betas](#)

Parameter of the exponential distribution

Specification

Alias: euv_betas

Argument(s): REALLIST

Description

Specifies the list of β parameters to define the distributions of the exponential random variables. Length must match the other parameters and the number of exponential random variables.

7.4.13.2 initial_point

- [Keywords Area](#)
- [variables](#)
- [exponential_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.13.3 descriptors

- [Keywords Area](#)
- [variables](#)
- [exponential_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: euv_descriptors

Argument(s): STRINGLIST

Default: euv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.14 `beta_uncertain`

- [Keywords Area](#)
- [variables](#)
- [beta_uncertain](#)

Aleatory uncertain variable - beta

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no beta uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		alphas	First parameter of the beta distribution
	Required		betas	Second parameter of the beta distribution
	Required		lower_bounds	Specify minimum values
	Required		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The number of beta uncertain variables, the alpha and beta parameters, and the distribution upper and lower bounds are required specifications, while the variable descriptors is an optional specification. The beta distribution can be

helpful when the actual distribution of an uncertain variable is unknown, but the user has a good idea of the bounds, the mean, and the standard deviation of the uncertain variable. The density function for the beta distribution is

$$f(x) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{(x-L)^{\alpha-1}(U-x)^{\beta-1}}{(U-L)^{\alpha+\beta-1}}$$

where $\Gamma(\alpha)$ is the gamma function and $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ is the beta function. To calculate the mean and standard deviation from the alpha, beta, upper bound, and lower bound parameters of the beta distribution, the following expressions may be used.

$$\mu = L + \frac{\alpha}{\alpha + \beta}(U - L)$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}(U - L)^2$$

Solving these for α and β gives:

$$\alpha = (\mu - L) \frac{(\mu - L)(U - \mu) - \sigma^2}{\sigma^2(U - L)}$$

$$\beta = (U - \mu) \frac{(\mu - L)(U - \mu) - \sigma^2}{\sigma^2(U - L)}$$

Note that the uniform distribution is a special case of this distribution for parameters $\alpha = \beta = 1$.

Theory

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means.

7.4.14.1 alphas

- [Keywords Area](#)
- [variables](#)
- [beta_uncertain](#)
- [alphas](#)

First parameter of the beta distribution

Specification

Alias: buv_alphas

Argument(s): REALLIST

Description

Specifies the list of α parameters to define the distributions of the beta random variables. Length must match the other parameters and the number of beta random variables.

7.4.14.2 betas

- [Keywords Area](#)
- [variables](#)
- [beta_uncertain](#)
- [betas](#)

Second parameter of the beta distribution

Specification

Alias: buv_betas

Argument(s): REALLIST

Description

Specifies the list of β parameters to define the distributions of the beta random variables. Length must match the other parameters and the number of beta random variables.

7.4.14.3 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [beta_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: buv_lower_bounds

Argument(s): REALLIST

Description

Specify minimum values

7.4.14.4 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [beta_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: `buv_upper_bounds`

Argument(s): REALLIST

Description

Specify maximum values

7.4.14.5 `initial_point`

- [Keywords Area](#)
- [variables](#)
- [beta_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.14.6 `descriptors`

- [Keywords Area](#)
- [variables](#)

- [beta_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: buv_descriptors

Argument(s): STRINGLIST

Default: buv_{}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.15 `gamma_uncertain`

- [Keywords Area](#)
- [variables](#)
- [gamma_uncertain](#)

Aleatory uncertain variable - gamma

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no gamma uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		alphas	First parameter of the gamma distribution

	Required	betas	Second parameter of the gamma distribution
	Optional	initial_point	Initial values for variables
	Optional	descriptors	Labels for the variables

Description

The gamma distribution is sometimes used to model time to complete a task, such as a repair or service task. It is a very flexible distribution with its shape governed by alpha and beta.

The density function for the gamma distribution is given by:

$$f(x) = \frac{x^{\alpha-1} \exp\left(\frac{-x}{\beta}\right)}{\beta^{\alpha} \Gamma(\alpha)},$$

$\mu = \alpha\beta$, and $\sigma^2 = \alpha\beta^2$. Note that the exponential distribution is a special case of this distribution for parameter $\alpha = 1$.

Theory

When used with some methods such as design of experiments and multidimensional parameter studies, distribution bounds are inferred to be $[0, \mu + 3\sigma]$.

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means.

7.4.15.1 alphas

- [Keywords Area](#)
- [variables](#)
- [gamma_uncertain](#)
- [alphas](#)

First parameter of the gamma distribution

Specification

Alias: gauv_alphas

Argument(s): REALLIST

Description

Specifies the list of α parameters to define the distributions of the gamma random variables. Length must match the other parameters and the number of gamma random variables.

7.4.15.2 betas

- [Keywords Area](#)
- [variables](#)
- [gamma_uncertain](#)
- [betas](#)

Second parameter of the gamma distribution

Specification

Alias: gauv_betas

Argument(s): REALLIST

Description

Specifies the list of β parameters to define the distributions of the gamma random variables. Length must match the other parameters and the number of gamma random variables.

7.4.15.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [gamma_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.15.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [gamma_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: `gauv_descriptors`

Argument(s): STRINGLIST

Default: `gauv_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.16 gumbel_uncertain

- [Keywords Area](#)
- [variables](#)
- [gumbel_uncertain](#)

Aleatory uncertain variable - gumbel

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification**Alias:** none**Argument(s):** INTEGER**Default:** no gumbel uncertain variables**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		alphas	First parameter of the gumbel distribution
	Required		betas	Second parameter of the gumbel distribution
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The Gumbel distribution is also referred to as the Type I Largest Extreme Value distribution. The distribution of maxima in sample sets from a population with a normal distribution will asymptotically converge to this distribution. It is commonly used to model demand variables such as wind loads and flood levels.

The density function for the Gumbel distribution is given by:

$$f(x) = \alpha \exp(-\alpha(x - \beta)) \exp(-e^{-\alpha(x - \beta)}),$$

$$\mu = \beta + \frac{0.5772}{\alpha}, \text{ and } \sigma = \frac{\pi}{\sqrt{6}\alpha}.$$

Theory

When used with some methods such as design of experiments and multidimensional parameter studies, distribution bounds are inferred to be $[\mu - 3\sigma, \mu + 3\sigma]$

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means.

7.4.16.1 alphas

- [Keywords Area](#)
- [variables](#)
- [gumbel_uncertain](#)
- [alphas](#)

First parameter of the gumbel distribution

Specification

Alias: guuv_alphas

Argument(s): REALLIST

Description

Specifies the list of β parameters to define the distributions of the gumbel random variables. Length must match the other parameters and the number of gumbel random variables.

7.4.16.2 betas

- [Keywords Area](#)
- [variables](#)
- [gumbel_uncertain](#)
- [betas](#)

Second parameter of the gumbel distribution

Specification

Alias: guuv_betas

Argument(s): REALLIST

Description

Specifies the list of β parameters to define the distributions of the gumbel random variables. Length must match the other parameters and the number of gumbel random variables.

7.4.16.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [gumbel_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.16.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [gumbel_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: guuv_descriptors

Argument(s): STRINGLIST

Default: guuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.17 frechet_uncertain

- [Keywords Area](#)
- [variables](#)
- [frechet_uncertain](#)

Aleatory uncertain variable - Frechet

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification**Alias:** none**Argument(s):** INTEGER**Default:** no frechet uncertain variables**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		alphas	First parameter of the Frechet distribution
	Required		betas	Second parameter of the Frechet distribution
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The Frechet distribution is also referred to as the Type II Largest Extreme Value distribution. The distribution of maxima in sample sets from a population with a lognormal distribution will asymptotically converge to this distribution. It is commonly used to model non-negative demand variables.

The density function for the frechet distribution is:

$$f(x) = \frac{\alpha}{\beta} \left(\frac{\beta}{x}\right)^{\alpha+1} \exp\left(-\left(\frac{\beta}{x}\right)^{\alpha}\right),$$

$$\mu = \beta\Gamma\left(1 - \frac{1}{\alpha}\right), \text{ and } \sigma^2 = \beta^2[\Gamma\left(1 - \frac{2}{\alpha}\right) - \Gamma^2\left(1 - \frac{1}{\alpha}\right)]$$

Theory

When used with some methods such as design of experiments and multidimensional parameter studies, distribution bounds are inferred to be $[0, \mu + 3\sigma]$.

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means.

7.4.17.1 alphas

- [Keywords Area](#)
- [variables](#)
- [frechet_uncertain](#)
- [alphas](#)

First parameter of the Frechet distribution

Specification

Alias: fuv_alphas

Argument(s): REALLIST

Description

Specifies the list of α parameters to define the distributions of the Frechet random variables. Length must match the other parameters and the number of Frechet random variables.

7.4.17.2 betas

- [Keywords Area](#)
- [variables](#)
- [frechet_uncertain](#)
- [betas](#)

Second parameter of the Frechet distribution

Specification

Alias: fuv_betas

Argument(s): REALLIST

Description

Specifies the list of β parameters to define the distributions of the Frechet random variables. Length must match the other parameters and the number of Frechet random variables.

7.4.17.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [frechet_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.17.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [frechet_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: fuv_descriptors

Argument(s): STRINGLIST

Default: fuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.18 weibull_uncertain

- [Keywords Area](#)
- [variables](#)
- [weibull_uncertain](#)

Aleatory uncertain variable - Weibull

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification**Alias:** none**Argument(s):** INTEGER**Default:** no weibull uncertain variables**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		alphas	First parameter of the Weibull distribution
	Required		betas	Second parameter of the Weibull distribution
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The Weibull distribution is also referred to as the Type III Smallest Extreme Value distribution. The Weibull distribution is commonly used in reliability studies to predict the lifetime of a device. It is also used to model capacity variables such as material strength.

The density function for the Weibull distribution is given by:

$$f(x) = \frac{\alpha}{\beta} \left(\frac{x}{\beta}\right)^{\alpha-1} \exp\left(-\left(\frac{x}{\beta}\right)^{\alpha}\right),$$

$$\mu = \beta\Gamma\left(1 + \frac{1}{\alpha}\right), \text{ and } \sigma = \mu\sqrt{\frac{\Gamma(1+\frac{2}{\alpha})}{\Gamma^2(1+\frac{1}{\alpha})} - 1}$$

7.4.18.1 alphas

- [Keywords Area](#)
- [variables](#)
- [weibull_uncertain](#)
- [alphas](#)

First parameter of the Weibull distribution

Specification

Alias: wuv_alphas

Argument(s): REALLIST

Description

Specifies the list of α parameters to define the distributions of the Weibull random variables.

Length must match the other parameters and the number of Weibull random variables.

7.4.18.2 betas

- [Keywords Area](#)

- [variables](#)
- [weibull_uncertain](#)
- [betas](#)

Second parameter of the Weibull distribution

Specification

Alias: wuv_betas

Argument(s): REALLIST

Description

Specifies the list of β parameters to define the distributions of the Weibull random variables. Length must match the other parameters and the number of Weibull random variables.

7.4.18.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [weibull_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.18.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [weibull_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: wuv_descriptors

Argument(s): STRINGLIST

Default: wuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.19 histogram_bin_uncertain

- [Keywords Area](#)
- [variables](#)
- [histogram_bin_uncertain](#)

Aleatory uncertain variable - continuous histogram

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no histogram bin uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		pairs_per_variable	Number of pairs defining each histogram bin variable
	Required		abscissas	Real abscissas for a bin histogram
	Required (<i>Choose One</i>)	Density Values (Group 1)	ordinates	Ordinates specifying a "skyline" probability density function
			counts	Frequency or relative probability of each bin
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

Histogram uncertain variables are typically used to model a set of empirical data. The bin histogram (contrast: [histogram_point_uncertain](#)) is a continuous aleatory distribution characterized by bins of non-zero width where the uncertain variable may lie, together with the relative frequencies of each bin. Hence it can be used to specify a marginal probability density function arising from data.

The `histogram_bin_uncertain` keyword specifies the number of variables to be characterized as continuous histograms. The required sub-keywords are: [abscissas](#) (ranges of values the variable can take on) and either [ordinates](#) or [counts](#) (characterizing each variable's frequency information). When using histogram bin variables, each variable must be defined by at least one bin (with two bounding value pairs). When more than one histogram bin variable is active, [pairs_per_variable](#) can be used to specify unequal apportionment of provided bin pairs among the variables.

The `abscissas` specification defines abscissa values ("x" coordinates) for the probability density function of each histogram variable. When paired with `counts`, the specifications provide sets of (x,c) pairs for each histogram variable where `c` defines a count (i.e., a frequency or relative probability) associated with a bin. If using bins of unequal width and specification of probability densities is more natural, then the `counts` specification can be replaced with an `ordinates` specification ("y" coordinates) in order to support interpretation of the input as (x,y) pairs defining the profile of a "skyline" probability density function.

Conversion between the two specifications is straightforward: a count/frequency is a cumulative probability quantity defined from the product of the ordinate density value and the `x` bin width. Thus, in the cases of bins of equal width, ordinate and count specifications are equivalent. In addition, `ordinates` and `counts` may be relative values; it is not necessary to scale them as all user inputs will be normalized.

To fully specify a bin-based histogram with `n` bins (potentially of unequal width), `n+1` (x,c) or (x,y) pairs must be specified with the following features:

- `x` is the parameter value for the left boundary of a histogram bin and `c` is the corresponding count for that bin. Alternatively, `y` defines the ordinate density value for this bin within a skyline probability density function. The right boundary of the bin is defined by the left boundary of the next pair.
- the final pair specifies the right end of the last bin and must have a `c` or `y` value of zero.

- the x values must be strictly increasing.
- all c or y values must be positive, except for the last which must be zero.
- a minimum of two pairs must be specified for each bin-based histogram variable.

Examples

The `pairs_per_variable` specification provides for the proper association of multiple sets of (x,c) or (x,y) pairs with individual histogram variables. For example, in this input snippet

```

histogram_bin_uncertain = 2
pairs_per_variable = 3      4
abscissas      = 5  8  10   .1 .2 .3 .4
counts         = 17 21  0   12 24 12 0
descriptors    = 'hbu_1'   'hbu_2'
```

`pairs_per_variable` associates the first 3 (x,c) pairs from `abscissas` and `counts` {(5,17),(8,21),(10,0)} with one bin-based histogram variable, where one bin is defined between 5 and 8 with a count of 17 and another bin is defined between 8 and 10 with a count of 21. The following set of 4 (x,c) pairs {(1,12),(2,24),(3,12),(4,0)} defines a second bin-based histogram variable containing three equal-width bins with counts 12, 24, and 12 (middle bin is twice as probable as the other two).

See Also

These keywords may also be of interest:

- [histogram_point_uncertain](#)

FAQ

Difference between bin and point histograms: A (continuous) bin histogram specifies bins of non-zero width, whereas a (discrete) point histogram specifies individual point values, which can be thought of as bins with zero width. In the terminology of LHS [93], the bin pairs specification defines a "continuous linear" distribution and the point pairs specification defines a "discrete histogram" distribution (although the points are real-valued, the number of possible values is finite).

7.4.19.1 pairs_per_variable

- [Keywords Area](#)
- [variables](#)
- [histogram_bin_uncertain](#)
- [pairs_per_variable](#)

Number of pairs defining each histogram bin variable

Specification

Alias: num_pairs

Argument(s): INTEGERLIST

Default: equal distribution

Description

By default, the list of `abscissas` and `counts` or `ordinates` will be evenly divided among the `histogram_bin_uncertain` variables. `pairs_per_variable` is a list of integers that specify the number of pairs to apportion to each variable.

7.4.19.2 `abscissas`

- [Keywords Area](#)
- [variables](#)
- [histogram_bin_uncertain](#)
- [abscissas](#)

Real abscissas for a bin histogram

Specification

Alias: `huv_bin_abscissas`

Argument(s): REALLIST

Description

A list of real abscissa ("x" coordinate) values characterizing the probability density function for each of the `histogram_bin_uncertain` variables. These are paired with either `counts` or `ordinates`. See [histogram_bin_uncertain](#) for details and examples.

7.4.19.3 `ordinates`

- [Keywords Area](#)
- [variables](#)
- [histogram_bin_uncertain](#)
- [ordinates](#)

Ordinates specifying a "skyline" probability density function

Specification

Alias: `huv_bin_ordinates`

Argument(s): REALLIST

Description

The `ordinates` list of real values defines the profile of a "skyline" probability density function by pairing with the specified `abscissas`. See [histogram_bin_uncertain](#) for details.

7.4.19.4 counts

- [Keywords Area](#)
- [variables](#)
- [histogram_bin_uncertain](#)
- [counts](#)

Frequency or relative probability of each bin

Specification

Alias: `huv_bin_counts`

Argument(s): REALLIST

Description

The `counts` list of real values gives the frequency or relative probability for each bin in a `histogram_bin_uncertain` specification. These are paired with the specified `abscissas`. See [histogram_bin_uncertain](#) for details.

7.4.19.5 initial_point

- [Keywords Area](#)
- [variables](#)
- [histogram_bin_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.19.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [histogram_bin_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: `huv_bin_descriptors`

Argument(s): STRINGLIST

Default: `hbu_v_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.20 poisson_uncertain

- [Keywords Area](#)
- [variables](#)
- [poisson_uncertain](#)

Aleatory uncertain discrete variable - Poisson

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [aleatory_uncertain_variables](#)

Specification**Alias:** none**Argument(s):** INTEGER**Default:** no poisson uncertain variables**Child Keywords:**

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		lambdas	The parameter for the Poisson distribution, the expected number of events in the time interval of interest
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The Poisson distribution is used to predict the number of discrete events that happen in a single time interval. The random events occur uniformly and independently. The expected number of occurrences in a single time interval is λ , which must be a positive real number. For example, if events occur on average 4 times per year and we are interested in the distribution of events over six months, λ would be 2. However, if we were interested in the distribution of events occurring over 5 years, λ would be 20.

The probability mass function for the poisson distribution is given by:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where

- λ is the expected number of events occurring in a single time interval
- x is the number of events that occur in this time period
- $f(x)$ is the probability that x events occur in this time period

Theory

When used with some methods such as design of experiments and multidimensional parameter studies, distribution bounds are inferred to be $[0, \mu + 3\sigma]$.

For some methods, including vector and centered parameter studies, an initial point is needed for the uncertain variables. When not given explicitly, these variables are initialized to their means.

7.4.20.1 lambdas

- [Keywords Area](#)
- [variables](#)
- [poisson_uncertain](#)
- [lambdas](#)

The parameter for the Poisson distribution, the expected number of events in the time interval of interest

Specification

Alias: none

Argument(s): REALLIST

Description

The density function for the poisson distribution is given by:

$$f(x) = \frac{\lambda e^{-\lambda}}{x!}$$

where λ is the frequency of events happening, and x is the number of events that occur.

7.4.20.2 initial_point

- [Keywords Area](#)
- [variables](#)
- [poisson_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.20.3 descriptors

- [Keywords Area](#)
- [variables](#)
- [poisson_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: puv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.21 binomial_uncertain

- [Keywords Area](#)
- [variables](#)
- [binomial_uncertain](#)

Aleatory uncertain discrete variable - binomial

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no binomial uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		probability_per_trial	A distribution parameter for the binomial distribution
	Required		num_trials	A distribution parameter
	Optional		initial_point	Initial values for variables

	Optional	descriptors	Labels for the variables
--	-----------------	-----------------------------	--------------------------

Description

The binomial distribution describes probabilities associated with a series of independent Bernoulli trials. A Bernoulli trial is an event with two mutually exclusive outcomes, such as 0 or 1, yes or no, success or fail. The probability of success remains the same (the trials are independent).

The density function for the binomial distribution is given by:

$$f(x) = \binom{n}{x} p^x (1-p)^{(n-x)}$$

where p is the probability of failure per trial, n is the number of trials and x is the number of successes.

Theory

The binomial distribution is typically used to predict the number of failures or defective items in a total of n independent tests or trials, where each trial has the probability p of failing or being defective.

7.4.21.1 probability_per_trial

- [Keywords Area](#)
- [variables](#)
- [binomial_uncertain](#)
- [probability_per_trial](#)

A distribution parameter for the binomial distribution

Specification

Alias: prob_per_trial

Argument(s): REALLIST

Description

The binomial distribution is typically used to predict the number of failures (or defective items or some type of event) in a total of n independent tests or trials, where each trial has the probability p of failing or being defective. Each particular test can be considered as a Bernoulli trial.

7.4.21.2 num_trials

- [Keywords Area](#)
- [variables](#)
- [binomial_uncertain](#)
- [num_trials](#)

A distribution parameter

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The binomial distribution is typically used to predict the number of failures (or defective items or some type of event) in a total of n independent tests or trials, where each trial has the probability p of failing or being defective. Each particular test can be considered as a Bernoulli trial.

7.4.21.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [binomial_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.21.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [binomial_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: biuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.22 `negative_binomial_uncertain`

- [Keywords Area](#)
- [variables](#)
- [negative_binomial_uncertain](#)

Aleatory uncertain discrete variable - negative binomial

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no negative binomial uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		probability_per_trial	A negative binomial distribution parameter
	Required		num_trials	A negative binomial distribution parameter
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The density function for the negative binomial distribution is given by:

$$f(x) = \binom{n+x-1}{x} p^n (1-p)^x$$

where

- p is the probability of success per trial
- n is the number of successful trials
- X is the number of failures

Theory

The negative binomial distribution is typically used to predict the number of failures observed when repeating a test until a total of n successes have occurred, where each test has a probability p of success.

7.4.22.1 [probability_per_trial](#)

- [Keywords Area](#)
- [variables](#)
- [negative_binomial_uncertain](#)
- [probability_per_trial](#)

A negative binomial distribution parameter

Specification

Alias: `prob_per_trial`

Argument(s): REALLIST

Description

The negative binomial distribution is typically used to predict the number of failures observed when repeating a test until a total of n successes have occurred, where each test has a probability p of success.

The density function for the negative binomial distribution is given by:

$$f(x) = \binom{n+x-1}{x} p^n (1-p)^x$$

where

- p is the probability of success per trial
- n is the number of successful trials
- X is the number of failures

7.4.22.2 num_trials

- [Keywords Area](#)
- [variables](#)
- [negative_binomial_uncertain](#)
- [num_trials](#)

A negative binomial distribution parameter

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The negative binomial distribution is typically used to predict the number of failures observed when repeating a test until a total of n successes have occurred, where each test has a probability p of success.

The density function for the negative binomial distribution is given by:

$$f(x) = \binom{n+x-1}{x} p^n (1-p)^x$$

where

- p is the probability of success per trial
- n is the number of successful trials
- X is the number of failures

7.4.22.3 initial_point

- [Keywords Area](#)
- [variables](#)
- [negative_binomial_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.22.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [negative_binomial_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: nbuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.23 `geometric_uncertain`

- [Keywords Area](#)
- [variables](#)
- [geometric_uncertain](#)

Aleatory uncertain discrete variable - geometric

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no geometric uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		probability_per_trial	Geometric distribution parameter
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The geometric distribution represents the number of successful trials that might occur before a failure is observed.

The density function for the geometric distribution is given by:

$$f(x) = p(1 - p)^x$$

where p is the probability of failure per trial.

7.4.23.1 `probability_per_trial`

- [Keywords Area](#)
- [variables](#)
- [geometric_uncertain](#)
- [probability_per_trial](#)

Geometric distribution parameter

Specification

Alias: prob_per_trial

Argument(s): REALLIST

Description

The geometric distribution represents the number of successful trials that occur before a failure is observed.

The density function for the geometric distribution is given by:

$$f(x) = p(1 - p)^x$$

where p is the probability of failure per trial and x is the number of successful trials.

7.4.23.2 initial_point

- [Keywords Area](#)
- [variables](#)
- [geometric_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.23.3 descriptors

- [Keywords Area](#)
- [variables](#)
- [geometric_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: `geuv_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.24 hypergeometric_uncertain

- [Keywords Area](#)
- [variables](#)
- [hypergeometric_uncertain](#)

Aleatory uncertain discrete variable - hypergeometric

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no hypergeometric uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		total_population	Parameter for the hypergeometric probability distribution describing the size of the total population
	Required		selected_population	Distribution parameter for the hypergeometric distribution describing the size of the population subset of interest
	Required		num_drawn	Distribution parameter for the hypergeometric distribution describing the number of draws from a combined population
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

The hypergeometric probability density is used when sampling without replacement from a total population of elements where

- The resulting element of each sample can be separated into one of two non-overlapping sets
- The probability of success changes with each sample.

The density function for the hypergeometric distribution is given by:

$$f(x) = \frac{\binom{m}{x} \binom{N-m}{n-x}}{\binom{N}{n}}$$

where the three distribution parameters are:

- N is the total population
- m is the number of items in the selected population (e.g. the number of white balls in the full urn of N items)
- n is the size of the sample drawn (e.g. number of balls drawn)

In addition,

- x, the abscissa of the density function, indicates the number of successes (e.g. drawing a white ball)
- $\binom{a}{b}$ indicates a binomial coefficient ("a choose b")

Theory

The hypergeometric is often described using an urn model. For example, say we have a total population containing N balls, and we know that m of the balls are white and the remaining balls are green. If we draw n balls from the urn without replacement, the hypergeometric distribution describes the probability of drawing x white balls.

7.4.24.1 total_population

- [Keywords Area](#)
- [variables](#)
- [hypergeometric_uncertain](#)
- [total_population](#)

Parameter for the hypergeometric probability distribution describing the size of the total population

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The density function for the hypergeometric distribution is given by:

$$f(x) = \frac{\binom{m}{x} \binom{N-m}{n-x}}{\binom{N}{n}}$$

where the three distribution parameters are:

- N is the total population
- m is the number of items in the selected population (e.g. the number of white balls in the full urn of N items)
- n is the size of the sample drawn (e.g. number of balls drawn)

In addition,

- x , the abscissa of the density function, indicates the number of successes (e.g. drawing a white ball)
- $\binom{a}{b}$ indicates a binomial coefficient ("a choose b")

7.4.24.2 selected_population

- [Keywords Area](#)
- [variables](#)
- [hypergeometric_uncertain](#)
- [selected_population](#)

Distribution parameter for the hypergeometric distribution describing the size of the population subset of interest

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The density function for the hypergeometric distribution is given by:

$$f(x) = \frac{\binom{m}{x} \binom{N-m}{n-x}}{\binom{N}{n}}$$

where the three distribution parameters are:

- N is the total population
- *m* is the number of items in the selected population (e.g. the number of white balls in the full urn of N items)
- n is the size of the sample drawn (e.g. number of balls drawn)

In addition,

- x, the abscissa of the density function, indicates the number of successes (e.g. drawing a white ball)
- $\binom{a}{b}$ indicates a binomial coefficient ("a choose b")

7.4.24.3 num_drawn

- [Keywords Area](#)
- [variables](#)
- [hypergeometric_uncertain](#)
- [num_drawn](#)

Distribution parameter for the hypergeometric distribution describing the number of draws from a combined population

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The density function for the hypergeometric distribution is given by:

$$f(x) = \frac{\binom{m}{x} \binom{N-m}{n-x}}{\binom{N}{n}}$$

where the three distribution parameters are:

- N is the total population
- m is the number of items in the selected population (e.g. the number of white balls in the full urn of N items)
- *n* is the size of the sample drawn (e.g. number of balls drawn)

In addition,

- x, the abscissa of the density function, indicates the number of successes (e.g. drawing a white ball)
- $\binom{a}{b}$ indicates a binomial coefficient ("a choose b")

7.4.24.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [hypergeometric_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.24.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [hypergeometric_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: hguv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.25 histogram_point_uncertain

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)

Aleatory uncertain variable - discrete histogram

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [aleatory_uncertain_variables](#)

Specification

Alias: none

Argument(s): none

Default: no histogram point uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		integer	Integer valued point histogram variable
	Optional		string	String (categorical) valued point histogram variable
	Optional		real	Real valued point histogram variable

Description

Histogram uncertain variables are typically used to model a set of empirical data. When the variables take on only discrete values or categories, a discrete, or point histogram is used to describe their probability mass function (one could think of this as a [histogram_bin_uncertain](#) variable with "bins" of zero width). Dakota supports integer-, string-, and real-valued point histograms.

Point histograms are similar to [discrete_design_set](#) and [discrete_state_set](#), but as they are uncertain variables, include the relative probabilities of observing the different values within the set.

The `histogram_point_uncertain` keyword is followed by one or more of `integer`, `string`, or `real`, each of which specify the number of variables to be characterized as discrete histograms of that sub-type.

Each discrete histogram variable is specified by one or more abscissa/count pairs. The `abscissas`, are the possible values the variable can take on ("x" coordinates of type integer, string, or real), and must be specified in increasing order. These are paired with `counts c` which provide the frequency of the given value or string, relative to other possible values/strings.

Thus, to fully specify a point-based histogram with n points, n (x,c) pairs must be specified with the following features:

- x is the point value (integer, string, or real) and c is the corresponding count for that value.
- the x values must be strictly increasing (lexicographically for strings).
- all c values must be positive.
- a minimum of one pair must be specified for each point-based histogram.

Examples

The `pairs_per_variable` specification provides for the proper association of multiple sets of (x,c) or (x,y) pairs with individual histogram variables. For example, in the following specification,

```

histogram_point_uncertain
  integer           = 2
  pairs_per_variable = 2      3
  abscissas         = 3 4    100 200 300
  counts            = 1 1    1  2  1

```

`pairs_per_variable` associates the (x,c) pairs {(3,1),(4,1)} with one point-based histogram variable (where the values 3 and 4 are equally probable) and associates the (x,c) pairs {(100,1),(200,2),(300,1)} with a second point-based histogram variable (where the value 200 is twice as probable as either 100 or 300).

See Also

These keywords may also be of interest:

- [histogram_bin_uncertain](#)

FAQ

Difference between bin and point histograms: A (continuous) bin histogram specifies bins of non-zero width, whereas a (discrete) point histogram specifies individual point values, which can be thought of as bins with zero width. In the terminology of LHS [93], the bin pairs specification defines a "continuous linear" distribution and the point pairs specification defines a "discrete histogram" distribution (although the points are real-valued, the number of possible values is finite).

7.4.25.1 integer

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [integer](#)

Integer valued point histogram variable

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		pairs_per_variable	Number of pairs defining each histogram point integer variable
	Required		abscissas	Integer abscissas for a point histogram
	Required		counts	Counts for integer-valued point histogram
	Optional		initial_point	Initial values for variables

	Optional	descriptors	Labels for the variables
--	-----------------	-----------------------------	--------------------------

Description

This probability mass function is integer-valued; the abscissa values must all be integers. The n abscissa values are paired with n `counts` which indicate the relative frequency (mass) of each integer relative to the other specified integers.

Examples

```

histogram_point_uncertain
  integer = 2
  pairs_per_variable = 2      3
  abscissas      = 3 4    100 200 300
  counts         = 1 1    1   2   1

```

There are two variables, the first one has two possible integer values which are equally probable. The second one has three options, and 200 is twice as probable as either 100 or 300.

7.4.25.1.1 `pairs_per_variable`

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [integer](#)
- [pairs_per_variable](#)

Number of pairs defining each histogram point integer variable

Specification

Alias: `num_pairs`

Argument(s): INTEGERLIST

Default: equal distribution

Description

By default, the list of `abscissas` and `counts` will be evenly divided among the histogram point integer variables. The number of `pairs_per_variable` specifies the apportionment of abscissa/count pairs among the histogram point integer variables. It must specify one integer ≥ 1 per variable that indicates how many of the (abscissa, count) = (x,c) pairs to associate with that variable.

7.4.25.1.2 `abscissas`

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [integer](#)

- [abscissas](#)

Integer abscissas for a point histogram

Specification

Alias: none

Argument(s): INTEGERLIST

Description

A list of integer abscissa ("x" coordinate) values characterizing the probability density function for each of the integer `histogram_point_uncertain` variables. These must be listed in increasing order for each variable, and are paired with [counts](#). See [histogram_point_uncertain](#) for details and examples.

7.4.25.1.3 counts

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [integer](#)
- [counts](#)

Counts for integer-valued point histogram

Specification

Alias: none

Argument(s): REALLIST

Description

Count or frequency for each of [abscissas](#). See [histogram_point_uncertain](#) for details and examples.

7.4.25.1.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [integer](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.25.1.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [integer](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: hpiv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.25.2 string

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [string](#)

String (categorical) valued point histogram variable

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		pairs_per_variable	Number of pairs defining each histogram point string variable
	Required		abscissas	String abscissas for a point histogram
	Required		counts	Counts for string-valued point histogram
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

This probability mass function is string-valued; the abscissa values must all be strings. The n abscissa values are paired with n counts which indicate the relative frequency (mass) of each string relative to the other specified strings.

Examples

```

histogram_point_uncertain
  string = 2
  pairs_per_variable = 2           3
  abscissas          = 'no' 'yes' 'function1' 'function2' 'function3'
  counts              = 1     1     1           2           1
  descriptors        = 'vote'    'which_function'
```

Here there are two variables, the first one ('vote') has two possible string values 'yes' and 'no' which are equally probable. The second one has three options for 'which_function', and 'function2' is twice as probable as 'function1' or 'function3'.

7.4.25.2.1 pairs_per_variable

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [string](#)
- [pairs_per_variable](#)

Number of pairs defining each histogram point string variable

Specification

Alias: num_pairs

Argument(s): INTEGERLIST

Default: equal distribution

Description

By default, the list of `abscissas` and `counts` will be evenly divided among the histogram point string variables. The number of `pairs_per_variable` specifies the apportionment of abscissa/count pairs among the histogram point string variables. It must specify one integer ≥ 1 per variable that indicates how many of the (abscissa, count) = (x,c) pairs to associate with that variable.

7.4.25.2.2 abscissas

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [string](#)
- [abscissas](#)

String abscissas for a point histogram

Specification

Alias: none

Argument(s): STRINGLIST

Description

A list of string abscissa ("x" coordinate) values characterizing the probability density function for each of the string `histogram_point_uncertain` variables. These must be listed in (lexicographically) increasing order for each variable, and are paired with `counts`. See [histogram_point_uncertain](#) for details and examples.

7.4.25.2.3 counts

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [string](#)
- [counts](#)

Counts for string-valued point histogram

Specification

Alias: none

Argument(s): REALLIST

Description

Count or frequency for each of [abscissas](#). See [histogram_point_uncertain](#) for details and examples.

7.4.25.2.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [string](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): STRINGLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.25.2.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [string](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: hpsv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.25.3 real

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [real](#)

Real valued point histogram variable

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		pairs_per_variable	Number of pairs defining each histogram point real variable
	Required		abscissas	Real abscissas for a point histogram
	Required		counts	Counts for real-valued point histogram
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

This probability mass function is real-valued; the abscissa values must all be integers. The n abscissa values are paired with n `counts` which indicate the relative frequency (mass) of each real relative to the other specified reals.

Examples

```

histogram_point_uncertain
  real = 2
  pairs_per_variable = 2          3
  abscissas          = 3.1415 4.5389 100 200.112345 300
  counts              = 1      1      1  2          1

```

There are two variables, the first one has two possible real values which are equally probable. The second one has three possible real value options, and 200.112345 is twice as probable as either 100 or 300.

7.4.25.3.1 pairs_per_variable

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [real](#)
- [pairs_per_variable](#)

Number of pairs defining each histogram point real variable

Specification

Alias: num_pairs

Argument(s): INTEGERLIST

Default: equal distribution

Description

By default, the list of `abscissas` and `counts` will be evenly divided among the histogram point real variables. The number of `pairs_per_variable` specifies the apportionment of abscissa/count pairs among the histogram point real variables. It must specify one integer ≥ 1 per variable that indicates how many of the (abscissa, count) = (x,c) pairs to associate with that variable.

7.4.25.3.2 abscissas

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [real](#)
- [abscissas](#)

Real abscissas for a point histogram

Specification

Alias: none

Argument(s): REALLIST

Description

A list of real abscissa ("x" coordinate) values characterizing the probability density function for each of the real `histogram_point_uncertain` variables. These must be listed in increasing order for each variable, and are paired with `counts`. See [histogram_point_uncertain](#) for details and examples.

7.4.25.3.3 counts

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [real](#)
- [counts](#)

Counts for real-valued point histogram

Specification

Alias: none

Argument(s): REALLIST

Description

Count or frequency for each of `abscissas`. See [histogram_point_uncertain](#) for details and examples.

7.4.25.3.4 initial_point

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [real](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.25.3.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [histogram_point_uncertain](#)
- [real](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: hpruv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.26 uncertain_correlation_matrix

- [Keywords Area](#)
- [variables](#)
- [uncertain_correlation_matrix](#)

Correlation among aleatory uncertain variables

Specification

Alias: none

Argument(s): REALLIST

Default: identity matrix (uncorrelated)

Description

Aleatory uncertain variables may have correlations specified through use of an `uncertain_correlation_matrix` specification. This specification is generalized in the sense that its specific meaning depends on the nondeterministic method in use.

When the method is a nondeterministic sampling method (i.e., [sampling](#)), then the correlation matrix specifies *rank correlations* [53].

When the method is a reliability (i.e., `local_reliability` or `global_reliability`) or stochastic expansion (i.e., `polynomial_chaos` or `stoch_collocation`) method, then the correlation matrix specifies *correlation coefficients* (normalized covariance) [43].

In either of these cases, specifying the identity matrix results in uncorrelated uncertain variables (the default). The matrix input should be symmetric and have all n^2 entries where n is the total number of aleatory uncertain variables.

Ordering of the aleatory uncertain variables is:

1. normal
2. lognormal
3. uniform
4. loguniform
5. triangular
6. exponential
7. beta
8. gamma
9. gumbel
10. frechet
11. weibull
12. histogram bin
13. poisson
14. binomial
15. negative binomial
16. geometric
17. hypergeometric
18. histogram point

When additional variable types are activated, they assume uniform distributions, and the ordering is as listed on [variables](#).

Examples

Consider the following random variables, distributions and correlations:

- X_1 , normal, uncorrelated with others
- X_2 , normal, correlated with X_3 , X_4 and X_5
- X_3 , weibull, correlated with X_5

- X_4 , exponential, correlated with X_3 , X_4 and X_5
- X_5 , normal, correlated with X_5 These correlations are captured by the following commands (order of the variables is respected).

```
uncertain_correlation_matrix
# ordering normal, exponential, weibull
# \f$X_1\f$ \f$X_2\f$ \f$X_5\f$ \f$X_4$f \f$X_3\f$
1.00 0.00 0.00 0.00 0.00
0.00 1.00 0.50 0.24 0.78
0.00 0.50 1.00 0.00 0.20
0.00 0.24 0.00 1.00 0.49
0.00 0.78 0.20 0.49 1.0
```

7.4.27 continuous_interval_uncertain

- [Keywords Area](#)
- [variables](#)
- [continuous_interval_uncertain](#)

Epistemic uncertain variable - values from one or more continuous intervals

Topics

This keyword is related to the topics:

- [continuous_variables](#)
- [epistemic_uncertain_variables](#)

Specification

Alias: interval_uncertain

Argument(s): INTEGER

Default: no continuous interval uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_intervals	Specify the number of intervals for each variable
	Optional		interval_probabilities	Assign probability mass to each interval
	Required		lower_bounds	Specify minimum values
	Required		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables

	Optional	descriptors	Labels for the variables
--	-----------------	--------------------	--------------------------

Description

Continuous interval uncertain variables are epistemic types. They can specify a single interval per variable which may be used in interval analysis, where the goal is to determine the interval bounds on the output corresponding to the interval bounds on the input. All values between the bounds are permissible. More detailed continuous interval representations can specify a set of belief structures based on intervals that may be contiguous, overlapping, or disjoint. This is used in specifying the inputs necessary for an epistemic uncertainty analysis using Dempster-Shafer theory of evidence.

Other epistemic types include:

- [discrete_interval_uncertain](#)
- [discrete_uncertain_set](#) [integer](#)
- [discrete_uncertain_set](#) [string](#)
- [discrete_uncertain_set](#) [real](#)

Examples

The following specification is for an interval analysis:

```
continuous_interval_uncertain = 2
lower_bounds = 2.0 4.0
upper_bounds = 2.5 5.0
```

The following specification is for a Dempster-Shafer analysis:

```
continuous_interval_uncertain = 2
num_intervals = 3 2
interval_probs = 0.25 0.5 0.25 0.4 0.6
lower_bounds = 2.0 4.0 4.5 1.0 3.0
upper_bounds = 2.5 5.0 6.0 5.0 5.0
```

Here there are 2 interval uncertain variables. The first one is defined by three intervals, and the second by two intervals. The three intervals for the first variable have basic probability assignments of 0.2, 0.5, and 0.3, respectively, while the basic probability assignments for the two intervals for the second variable are 0.4 and 0.6. The basic probability assignments for each interval variable must sum to one. The interval bounds for the first variable are [2, 2.5], [4, 5], and [4.5, 6], and the interval bounds for the second variable are [1.0, 5.0] and [3.0, 5.0]. Note that the intervals can be overlapping or disjoint. The BPA for the first variable indicates that it is twice as likely that the value occurs on the interval [4,5] than either [2,2.5] or [4.5,6].

Theory

The continuous interval uncertain variable is NOT a probability distribution. Although it may seem similar to a histogram, the interpretation of this uncertain variable is different. It is used in epistemic uncertainty analysis, where one is trying to model uncertainty due to lack of knowledge. The continuous interval uncertain variable is used in both interval analysis and in Dempster-Shafer theory of evidence.

- interval analysis -only one interval is allowed for each `continuous_interval_uncertain` variable
-the interval is defined by lower and upper bounds -the value of the random variable lies somewhere in this interval -output is the minimum and maximum function value conditional on the specified interval

- Dempster-Shafer theory of evidence -multiple intervals can be assigned to each `continuous_interval_uncertain` variable -a Basic Probability Assignment (BPA) is associated with each interval. The BPA represents a probability that the value of the uncertain variable is located within that interval. -each interval is defined by lower and upper bounds -outputs are called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the intervals together with their BPA.

7.4.27.1 `num_intervals`

- [Keywords Area](#)
- [variables](#)
- [continuous_interval_uncertain](#)
- [num_intervals](#)

Specify the number of intervals for each variable

Specification

Alias: `iuv_num_intervals`

Argument(s): INTEGERLIST

Default: Equal apportionment of intervals among variables

Description

In Dakota, epistemic uncertainty analysis is performed using either interval estimation or Dempster-Shafer theory of evidence. In these approaches, one does not assign a probability distribution to each uncertain input variable. Rather, one divides each uncertain input variable into one or more intervals. The input parameters are only known to occur within intervals; nothing more is assumed. `num_intervals` specifies the number of such intervals associated with each interval uncertain parameter.

7.4.27.2 `interval_probabilities`

- [Keywords Area](#)
- [variables](#)
- [continuous_interval_uncertain](#)
- [interval_probabilities](#)

Assign probability mass to each interval

Specification

Alias: `interval_probs` `iuv_interval_probs`

Argument(s): REALLIST

Default: Equal probability assignments for each interval ($1/\text{num_intervals}[i]$)

Description

The basic probability assignments for each interval variable must sum to one. For example, if an interval variable is defined with three intervals, the probabilities for these intervals could be 0.2, 0.5, and 0.3 which sum to one, but could not be 0.5,0.5, and 0.5 which do not sum to one.

7.4.27.3 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [continuous_interval_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: none

Argument(s): REALLIST

Description

Specify minimum values

7.4.27.4 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [continuous_interval_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: none

Argument(s): REALLIST

Description

Specify maximum values

7.4.27.5 initial_point

- [Keywords Area](#)
- [variables](#)
- [continuous_interval_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.27.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [continuous_interval_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: iuv_descriptors

Argument(s): STRINGLIST

Default: ciuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.28 `discrete_interval_uncertain`

- [Keywords Area](#)
- [variables](#)
- [discrete_interval_uncertain](#)

Epistemic uncertain variable - values from one or more discrete intervals

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [epistemic_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: No discrete interval uncertain variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		num_intervals	Specify the number of intervals for each variable
	Optional		interval_probabilities	Assign probability mass to each interval
	Required		lower_bounds	Specify minimum values
	Required		upper_bounds	Specify maximum values
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

Discrete interval uncertain variables are epistemic types. They can specify a single interval per variable which may be used in interval analysis, where the goal is to determine the interval bounds on the output corresponding to the

interval bounds on the input. Permissible values are any integer within the bound. More detailed continuous interval representations can specify a set of belief structures based on intervals that may be contiguous, overlapping, or disjoint. This is used in specifying the inputs necessary for an epistemic uncertainty analysis using Dempster-Shafer theory of evidence.

Other epistemic types include:

- [continuous_interval_uncertain](#)
- [discrete_uncertain_set](#) [integer](#)
- [discrete_uncertain_set](#) [string](#)
- [discrete_uncertain_set](#) [real](#)

Examples

Let d1 be 2, 3 or 4 with probability 0.2, 4 or 5 with probability 0.5 and 6 with probability 0.3. Let d2 be 4, 5 or 6 with probability 0.4 and 6, 7 or 8 with probability 0.6. The following specification is for a Dempster-Shafer analysis:

```
discrete_interval_uncertain = 2
num_intervals = 3 2
interval_probs = 0.2 0.5 0.3 0.4 0.6
lower_bounds = 2 4 6 4 6
upper_bounds = 4 5 6 6 8
```

Theory

- Dempster-Shafer theory of evidence -multiple intervals can be assigned to each `discrete_interval_uncertain` variable -a Basic Probability Assignment (BPA) is associated with each interval. The BPA represents a probability that the value of the uncertain variable is located within that interval. -each interval is defined by lower and upper bounds -outputs are called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the intervals together with their BPA.

7.4.28.1 num_intervals

- [Keywords Area](#)
- [variables](#)
- [discrete_interval_uncertain](#)
- [num_intervals](#)

Specify the number of intervals for each variable

Specification

Alias: none

Argument(s): INTEGERLIST

Default: Equal apportionment of intervals among variables

Description

In Dakota, epistemic uncertainty analysis is performed using either interval estimation or Dempster-Shafer theory of evidence. In these approaches, one does not assign a probability distribution to each uncertain input variable. Rather, one divides each uncertain input variable into one or more intervals. The input parameters are only known to occur within intervals; nothing more is assumed. `num_intervals` specifies the number of such intervals associated with each interval uncertain parameter.

7.4.28.2 `interval_probabilities`

- [Keywords Area](#)
- [variables](#)
- [discrete_interval_uncertain](#)
- [interval_probabilities](#)

Assign probability mass to each interval

Specification

Alias: `interval_probs` `range_probabilities` `range_probs`

Argument(s): REALLIST

Default: Equal probability assignments for each interval ($1/\text{num_intervals}[i]$)

Description

The basic probability assignments for each interval variable must sum to one. For example, if an interval variable is defined with three intervals, the probabilities for these intervals could be 0.2, 0.5, and 0.3 which sum to one, but could not be 0.5, 0.5, and 0.5 which do not sum to one.

7.4.28.3 `lower_bounds`

- [Keywords Area](#)
- [variables](#)
- [discrete_interval_uncertain](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specify minimum values

7.4.28.4 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [discrete_interval_uncertain](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specify maximum values

7.4.28.5 initial_point

- [Keywords Area](#)
- [variables](#)
- [discrete_interval_uncertain](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.28.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_interval_uncertain](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: diuv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.29 discrete_uncertain_set

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)

Epistemic uncertain variable - discrete set-valued

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [epistemic_uncertain_variables](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		integer	Discrete, epistemic uncertain variable - integers within a set
	Optional		string	Discrete, epistemic uncertain variable - strings within a set
	Optional		real	Discrete, epistemic uncertain variable - real numbers within a set

Description

Discrete uncertain variables whose values come from a set of admissible elements. Each variable specified must be of type integer, string, or real.

7.4.29.1 integer

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [integer](#)

Discrete, epistemic uncertain variable - integers within a set

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [epistemic_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete uncertain set integer variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_variable	Number of admissible elements for each set variable
	Required		elements	The permissible values for each discrete variable
	Optional		set_probabilities	This keyword defines the probabilities for the various elements of discrete sets.
	Optional		categorical	Whether the set-valued variables are categorical or relaxable
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

Discrete set variables may be used to specify categorical choices which are epistemic. For example, if we have three possible forms for a physics model (model 1, 2, or 3) and there is epistemic uncertainty about which one is correct, a discrete uncertain set may be used to represent this type of uncertainty.

This variable is defined by a set of integers, in which the discrete value may take any value within the integer set (for example, the set may be defined as 1, 2, and 4)

Other epistemic types include:

- [continuous_interval_uncertain](#)
- [discrete_interval_uncertain](#)
- [discrete_uncertain_set](#) [string](#)
- [discrete_uncertain_set](#) [real](#)

Examples

Let d1 be 2 or 13 and d2 be 4, 5 or 26. The following specification is for an interval analysis:

```
discrete_uncertain_set
  integer
  num_set_values 2    3
  set_values    2 13  4 5 26
  descriptors   'd1' 'd2'
```

Theory

The `discrete_uncertain_set-integer` variable is NOT a discrete random variable. It can be contrasted to a the histogram-defined random variables: [histogram_bin_uncertain](#) and [histogram_point_uncertain](#). It is used in epistemic uncertainty analysis, where one is trying to model uncertainty due to lack of knowledge.

The discrete uncertain set integer variable is used in both interval analysis and in Dempster-Shafer theory of evidence.

- interval analysis -the values are integers, equally weighted -the true value of the random variable is one of the integers in this set -output is the minimum and maximum function value conditional on the specified inputs
- Dempster-Shafer theory of evidence -the values are integers, but they can be assigned different weights -outputs are called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the values together with their weights.

7.4.29.1.1 `elements_per_variable`

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [integer](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: `num_set_values`

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.29.1.2 `elements`

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [integer](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): INTEGERLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.29.1.3 set_probabilities

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [integer](#)
- [set_probabilities](#)

This keyword defines the probabilities for the various elements of discrete sets.

Specification

Alias: set_probs

Argument(s): REALLIST

Default: Equal probability assignments for each set member (1/num_set_values[i])

Description

There are three types of `discrete_uncertain_set` variables: integer, string, or real sets. With each of these types, one defines the number of elements of the set per that variable, the values of those elements, and the associated probabilities. For example, if one has an integer discrete uncertain set variable with 3 elements {3,4,8}, then one could define the probabilities associated with those set elements as (for example) 0.2, 0.5, and 0.3. The `set_probabilities` for a particular variable should sum to one over all the elements in that set.

7.4.29.1.4 categorical

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [integer](#)
- [categorical](#)

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none

Argument(s): STRINGLIST

Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```
discrete_design_set
  integer 1
    elements 2 4 7
  descriptor 'rotor_blades'
  categorical 'no'
```

7.4.29.1.5 initial_point

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [integer](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.29.1.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [integer](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: `dusiv_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.29.2 string

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [string](#)

Discrete, epistemic uncertain variable - strings within a set

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [epistemic_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete uncertain set string variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_variable	Number of admissible elements for each set variable
	Required		elements	The permissible values for each discrete variable
	Optional		set_probabilities	This keyword defines the probabilities for the various elements of discrete sets.
	Optional		initial_point	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

Discrete set variables may be used to specify categorical choices which are epistemic. For example, if we have three possible forms for a physics model (model 1, 2, or 3) and there is epistemic uncertainty about which one is correct, a discrete uncertain set may be used to represent this type of uncertainty.

This variable is defined by a set of strings, in which the discrete value may take any value within the string set (for example, the set may be defined as 'coarse', 'medium', and 'fine')

Other epistemic types include:

- [continuous_interval_uncertain](#)
- [discrete_interval_uncertain](#)
- [discrete_uncertain_set integer](#)
- [discrete_uncertain_set real](#)

Examples

```
discrete_uncertain_set
string
num_set_values 2          3
set_values     'red' 'blue' 'coarse' 'medium' 'fine'
descriptors   'ds1'      'ds2'
```

7.4.29.2.1 elements_per_variable

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [string](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.29.2.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [string](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): STRINGLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.29.2.3 set_probabilities

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [string](#)
- [set_probabilities](#)

This keyword defines the probabilities for the various elements of discrete sets.

Specification

Alias: set_probs

Argument(s): REALLIST

Default: Equal probability assignments for each set member ($1/\text{num_set_values}[i]$)

Description

There are three types of `discrete_uncertain_set` variables: integer, string, or real sets. With each of these types, one defines the number of elements of the set per that variable, the values of those elements, and the associated probabilities. For example, if one has an integer discrete uncertain set variable with 3 elements {3,4,8}, then one could define the probabilities associated with those set elements as (for example) 0.2, 0.5, and 0.3. The `set_probabilities` for a particular variable should sum to one over all the elements in that set.

7.4.29.2.4 `initial_point`

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [string](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): STRINGLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.29.2.5 `descriptors`

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [string](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: `dussv_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.29.3 real

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [real](#)

Discrete, epistemic uncertain variable - real numbers within a set

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [epistemic_uncertain_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete uncertain set real variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_-variable	Number of admissible elements for each set variable

	Required	elements	The permissible values for each discrete variable
	Optional	set_probabilities	This keyword defines the probabilities for the various elements of discrete sets.
	Optional	categorical	Whether the set-valued variables are categorical or relaxable
	Optional	initial_point	Initial values for variables
	Optional	descriptors	Labels for the variables

Description

Discrete set variables may be used to specify categorical choices which are epistemic. For example, if we have three possible forms for a physics model (model 1, 2, or 3) and there is epistemic uncertainty about which one is correct, a discrete uncertain set may be used to represent this type of uncertainty.

This variable is defined by a set of reals, in which the discrete variable may take any value defined within the real set (for example, a parameter may have two allowable real values, 3.285 or 4.79).

Other epistemic types include:

- [continuous_interval_uncertain](#)
- [discrete_interval_uncertain](#)
- [discrete_uncertain_set](#) [integer](#)
- [discrete_uncertain_set](#) [string](#)

Examples

Let d1 be 2.1 or 1.3 and d2 be 0.4, 5 or 2.6. The following specification is for an interval analysis:

```
discrete_uncertain_set
integer
num_set_values 2          3
set_values     2.1  1.3   0.4  5   2.6
descriptors    'dr1'    'dr2'
```

Theory

The `discrete_uncertain_set-integer` variable is NOT a discrete random variable. It can be contrasted to the histogram-defined random variables: [histogram_bin_uncertain](#) and [histogram_point_uncertain](#). It is used in epistemic uncertainty analysis, where one is trying to model uncertainty due to lack of knowledge.

The discrete uncertain set integer variable is used in both interval analysis and in Dempster-Shafer theory of evidence.

- interval analysis -the values are integers, equally weighted -the true value of the random variable is one of the integers in this set -output is the minimum and maximum function value conditional on the specified inputs
- Dempster-Shafer theory of evidence -the values are integers, but they can be assigned different weights -outputs are called "belief" and "plausibility." Belief represents the smallest possible probability that is consistent with the evidence, while plausibility represents the largest possible probability that is consistent with the evidence. Evidence is the values together with their weights.

7.4.29.3.1 elements_per_variable

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [real](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.29.3.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [real](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): REALLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.29.3.3 set_probabilities

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [real](#)
- [set_probabilities](#)

This keyword defines the probabilities for the various elements of discrete sets.

Specification

Alias: set_probs

Argument(s): REALLIST

Default: Equal probability assignments for each set member (1/num_set_values[i])

Description

There are three types of `discrete_uncertain_set` variables: integer, string, or real sets. With each of these types, one defines the number of elements of the set per that variable, the values of those elements, and the associated probabilities. For example, if one has an integer discrete uncertain set variable with 3 elements {3,4,8}, then one could define the probabilities associated with those set elements as (for example) 0.2, 0.5, and 0.3. The `set_probabilities` for a particular variable should sum to one over all the elements in that set.

7.4.29.3.4 categorical

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [real](#)
- [categorical](#)

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none

Argument(s): STRINGLIST

Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in `[yYnNtTfF][.]*`

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```
discrete_design_set
  integer 1
    elements 2 4 7
  descriptor 'rotor_blades'
  categorical 'no'
```

7.4.29.3.5 initial_point

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [real](#)
- [initial_point](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.29.3.6 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_uncertain_set](#)
- [real](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: `dusrv_{i}`

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.30 continuous_state

- [Keywords Area](#)
- [variables](#)
- [continuous_state](#)

State variable - continuous

Topics

This keyword is related to the topics:

- [state_variables](#)
- [continuous_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: No continuous state variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_state	Initial values for variables
	Optional		lower_bounds	Specify minimum values
	Optional		upper_bounds	Specify maximum values
	Optional		descriptors	Labels for the variables

Description

Continuous state variables are defined by bounds.

Default behavior for most methods is that only the `initial_state` values are used.

See the [state_variables](#) page for details on the behavior of state variables.

7.4.30.1 `initial_state`

- [Keywords Area](#)
- [variables](#)
- [continuous_state](#)
- [initial_state](#)

Initial values for variables

Specification

Alias: `csv_initial_state`

Argument(s): REALLIST

Default: 0.0

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.30.2 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [continuous_state](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: csv_lower_bounds

Argument(s): REALLIST

Default: -infinity

Description

Specify minimum values

7.4.30.3 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [continuous_state](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: csv_upper_bounds

Argument(s): REALLIST

Default: infinity

Description

Specify maximum values

7.4.30.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [continuous_state](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: csv_descriptors

Argument(s): STRINGLIST

Default: csv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.31 discrete_state_range

- [Keywords Area](#)
- [variables](#)
- [discrete_state_range](#)

State variables - discrete range-valued

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [state_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: No discrete state variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		initial_state	Initial values for variables
	Optional		lower_bounds	Specify minimum values
	Optional		upper_bounds	Specify maximum values
	Optional		descriptors	Labels for the variables

Description

Discrete state variables defined by bounds (an integer interval).

The details of how to specify this discrete variable are located on the [discrete_variables](#) page.

See the [state_variables](#) page for details on the behavior of state variables.

7.4.31.1 [initial_state](#)

- [Keywords Area](#)
- [variables](#)
- [discrete_state_range](#)
- [initial_state](#)

Initial values for variables

Specification

Alias: `dsv_initial_state`

Argument(s): INTEGERLIST

Default: 0

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.31.2 lower_bounds

- [Keywords Area](#)
- [variables](#)
- [discrete_state_range](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: dsv_lower_bounds

Argument(s): INTEGERLIST

Default: INT_MIN

Description

Specify minimum values

7.4.31.3 upper_bounds

- [Keywords Area](#)
- [variables](#)
- [discrete_state_range](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: dsv_upper_bounds

Argument(s): INTEGERLIST

Default: INT_MAX

Description

Specify maximum values

7.4.31.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_state_range](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: dsv_descriptors

Argument(s): STRINGLIST

Default: dsriv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.32 discrete_state_set

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)

State variable - discrete set-valued

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [state_variables](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	integer	Discrete state variables, each defined by a set of permissible integers
	Optional	string	String-valued discrete state set variables
	Optional	real	Discrete state variables, each defined by a set of permissible real numbers

Description

Discrete state variables whose values come from a set of admissible elements. Each variable specified must be of type integer, string, or real.

7.4.32.1 integer

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [integer](#)

Discrete state variables, each defined by a set of permissible integers

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [state_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete state set integer variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_variable	Number of admissible elements for each set variable
	Required		elements	The permissible values for each discrete variable
	Optional		categorical	Whether the set-valued variables are categorical or relaxable
	Optional		initial_state	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

Discrete state variables defined by a set of permissible integers.

The details of how to specify this discrete variable are located on the [discrete_variables](#) page.

See the [state_variables](#) page for details on the behavior of state variables.

7.4.32.1.1 [elements_per_variable](#)

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [integer](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.32.1.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [integer](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): INTEGERLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.32.1.3 categorical

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [integer](#)
- [categorical](#)

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none

Argument(s): STRINGLIST

Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.

```
discrete_design_set
  integer 1
    elements 2 4 7
  descriptor 'rotor_blades'
  categorical 'no'
```

7.4.32.1.4 initial_state

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [integer](#)
- [initial_state](#)

Initial values for variables

Specification

Alias: none

Argument(s): INTEGERLIST

Default: middle set value, or rounded down

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.32.1.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [integer](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: dssiv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.32.2 string

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [string](#)

String-valued discrete state set variables

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [state_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete state set string variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_-variable	Number of admissible elements for each set variable

	Required	elements	The permissible values for each discrete variable
	Optional	initial_state	Initial values for variables
	Optional	descriptors	Labels for the variables

Description

Discrete state variables whose values come from a specified set of admissible strings. The details of how to specify this discrete variable are located on the [discrete_variables](#) page. See the [state_variables](#) page for details on the behavior of state variables. Each string element value must be quoted and may contain alphanumeric, dash, underscore, and colon. White space, quote characters, and backslash/metacharacters are not permitted.

7.4.32.2.1 elements_per_variable

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [string](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.32.2.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [string](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): STRINGLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.32.2.3 initial_state

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [string](#)
- [initial_state](#)

Initial values for variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: middle set value, or rounded down

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.32.2.4 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [string](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: dsssv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.32.3 real

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [real](#)

Discrete state variables, each defined by a set of permissible real numbers

Topics

This keyword is related to the topics:

- [discrete_variables](#)
- [state_variables](#)

Specification

Alias: none

Argument(s): INTEGER

Default: no discrete state set real variables

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		elements_per_variable	Number of admissible elements for each set variable
	Required		elements	The permissible values for each discrete variable
	Optional		categorical	Whether the set-valued variables are categorical or relaxable
	Optional		initial_state	Initial values for variables
	Optional		descriptors	Labels for the variables

Description

Discrete state variables defined by a set of permissible real numbers.

The details of how to specify this discrete variable are located on the [discrete_variables](#) page.

See the [state_variables](#) page for details on the behavior of state variables.

7.4.32.3.1 [elements_per_variable](#)

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [real](#)
- [elements_per_variable](#)

Number of admissible elements for each set variable

Specification

Alias: num_set_values

Argument(s): INTEGERLIST

Default: equal distribution

Description

Discrete set variables (including design, uncertain, and state) take on only a fixed set of values. For each type (integer, string, or real), this keyword specifies how many admissible values are provided for each variable. If not specified, equal apportionment of elements among variables is assumed, and the number of elements must be evenly divisible by the number of variables.

7.4.32.3.2 elements

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [real](#)
- [elements](#)

The permissible values for each discrete variable

Specification

Alias: set_values

Argument(s): REALLIST

Description

Specify the permissible values for discrete set variables (of type integer, string, or real). See the description on the [discrete_variables](#) page.

7.4.32.3.3 categorical

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [real](#)
- [categorical](#)

Whether the set-valued variables are categorical or relaxable

Specification

Alias: none

Argument(s): STRINGLIST

Description

A list of strings of length equal to the number of set (integer, string, or real) variables indicating whether they are strictly categorical, meaning may only take on values from the provided set, or relaxable, meaning may take on any integer or real value between the lowest and highest specified element. Valid categorical strings include 'yes', 'no', 'true', and 'false', or any abbreviation in [yYnNtTfF][.]*

Examples

Discrete_design_set variable, 'rotor_blades', can take on only integer values, 2, 4, or 7 by default. Since categorical is specified to be false, the integrality can be relaxed and 'rotor_blades' can take on any value between 2 and 7, e.g., 3, 6, or 5.5.


```
discrete_design_set
  integer 1
    elements 2 4 7
  descriptor 'rotor_blades'
  categorical 'no'
```

7.4.32.3.4 initial_state

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [real](#)
- [initial_state](#)

Initial values for variables

Specification

Alias: none

Argument(s): REALLIST

Default: middle set value, or rounded down

Description

Specifies the initial or default value a variable should assume, for example when starting a method or fixing the value of an inactive variable.

Default Behavior

Range variables, including [continuous_design](#), [continuous_state](#), [discrete_design_range](#), and [discrete_state_range](#) are initialized to:

- Mean/midpoint (or next smaller integer value) when doubly-bounded
- 0 when unbounded
- 0 when semi-bounded and zero is in the half interval, otherwise to the specified lower or upper bound

Set variables, including [discrete_design_set](#) and [discrete_state_set](#) default to the middle (or next smaller) value of the ordered set of admissible values.

Uncertain variables default initialize to their mean (or nearest admissible discrete) value, truncating to any specified lower or upper bounds as needed. Epistemic uncertain variable means are calculated making a probabilistic assumption on the specified intervals or set values.

7.4.32.3.5 descriptors

- [Keywords Area](#)
- [variables](#)
- [discrete_state_set](#)
- [real](#)
- [descriptors](#)

Labels for the variables

Specification

Alias: none

Argument(s): STRINGLIST

Default: dssrv_{i}

Description

The optional variable labels specification `descriptors` is a list of strings which identify the variables. These are used in console and tabular output.

The default descriptor strings use a variable type-dependent root string plus a numeric identifier.

7.4.33 `linear_inequality_constraint_matrix`

- [Keywords Area](#)
- [variables](#)
- [linear_inequality_constraint_matrix](#)

Define coefficients of the linear inequality constraints

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): REALLIST

Default: no linear inequality constraints

Description

In the inequality case, the constraint matrix A provides coefficients for the variables in the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where the bounds are optionally specified by `linear_inequality_lower_bounds`, and `linear_inequality_upper_bounds`. The bounds, if not specified, will default to -infinity, and 0, respectively, resulting in one-sided inequalities of the form

$$Ax \leq 0.0$$

The `linear_constraints` topics page (linked above) outlines a few additional things to consider when using linear constraints.

Examples

In the first example, an optimization involving two variables, x_1 and x_2 , is to be performed. These variables must satisfy two constraints:

$$\begin{aligned} 1.5 \cdot x_1 + 1.0 \cdot x_2 &\leq 5.0 \\ x_1 \leq x_2 &\implies x_1 - x_2 \leq 0.0 \end{aligned}$$

The pair of constraints can be written in matrix form as:

$$\begin{bmatrix} 1.5 & 1.0 \\ 1.0 & -1.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 5.0 \\ 0.0 \end{bmatrix}$$

The coefficient matrix and right hand side of this matrix inequality are expressed to Dakota in the variables section of the input file:

```
variables
  continuous_design 2
  descriptors 'x1' 'x2'

  linear_inequality_constraint_matrix = 1.5  1.0
                                       1.0 -1.0

  linear_inequality_upper_bounds = 5.0
                                   0.0
```

The second example is more complex in two respects. First, some, but not all, of the constraints are "two sided", with both lower and upper bounds. Second, not all variables participate in all constraints.

There are four variables, x_1 , x_2 , x_3 , and x_4 , and four constraints.

$$\begin{aligned} -2.0 &\leq 5.0 \cdot x_1 + 2.0 \cdot x_2 \leq 9.0 \\ 0.0 &\leq x_1 + x_3 \\ -8.0 &\leq x_2 + 6.0 \cdot x_4 \leq 8.0 \\ x_1 + x_2 + x_3 &\leq 9.0 \end{aligned}$$

Or, in matrix form,

$$\begin{bmatrix} -2.0 \\ 0.0 \\ -8.0 \\ -\infty \end{bmatrix} \leq \begin{bmatrix} 5.0 & 2.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 6.0 \\ 1.0 & 1.0 & 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \leq \begin{bmatrix} 9.0 \\ \infty \\ 8.0 \\ 9.0 \end{bmatrix}$$

The Dakota specification for this matrix inequality is:

```
variables
  continuous_design 4
  descriptors 'x1' 'x2' 'x3' 'x4'

  linear_inequality_constraint_matrix = 5.0  2.0  0.0  0.0
                                       1.0  0.0  1.0  0.0
                                       0.0  1.0  0.0  6.0
                                       1.0  1.0  1.0  0.0

  linear_inequality_lower_bounds = -2.0
                                   0.0
                                   -8.0
                                   -inf

  linear_inequality_upper_bounds = 9.0
                                   inf
                                   8.0
                                   9.0
```

7.4.34 `linear_inequality_lower_bounds`

- [Keywords Area](#)
- [variables](#)
- [linear_inequality_lower_bounds](#)

Define lower bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = -infinity

Description

In the inequality case, the lower a_l and upper a_u bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where A is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see [objective_functions](#)), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than `+bigRealBoundSize` (1.e+30, as defined in `Minimizer`) are treated as +infinity and any lower bound values less than `-bigRealBoundSize` are treated as -infinity.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since `-DBL_MAX < -bigRealBoundSize`).

Examples

Examples of specifying linear inequality constraints to Dakota are provided on the [linear_inequality_constraint_matrix](#) page.

7.4.35 `linear_inequality_upper_bounds`

- [Keywords Area](#)
- [variables](#)
- [linear_inequality_upper_bounds](#)

Define upper bounds for the linear inequality constraint

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0 .

Description

In the inequality case, the lower a_l and upper a_u bounds provide constraint limits for the two-sided formulation:

$$a_l \leq Ax \leq a_u$$

Where A is the constrain matrix of variable coefficients.

As with nonlinear inequality constraints (see [objective_functions](#)), the default linear inequality constraint bounds are selected so that one-sided inequalities of the form

$$Ax \leq 0.0$$

result when there are no user bounds specifications (this provides backwards compatibility with previous Dakota versions).

In a user bounds specification, any upper bound values greater than `+bigRealBoundSize` (1.e+30, as defined in `Minimizer`) are treated as `+infinity` and any lower bound values less than `-bigRealBoundSize` are treated as `-infinity`.

This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since `-DBL_MAX < -bigRealBoundSize`).

Examples

Examples of specifying linear inequality constraints to Dakota are provided on the [linear_inequality_constraint_matrix](#) page.

7.4.36 linear_inequality_scale_types

- [Keywords Area](#)
- [variables](#)
- [linear_inequality_scale_types](#)

How to scale each linear inequality constraint

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

Each string in `linear_inequality_scale_types` indicates the scaling type for each linear inequality constraint. They only have effect when the associated method specifies `scaling`.

The options are:

- 'value' - characteristic value. If this is chosen, then `linear_inequality_scales` must be specified; 'value' is assumed if scales are given without `scale_types`
- 'auto' - automatic scaling.

If a single string is specified it will apply to all linear inequality constraints. Otherwise the number of strings specified should be equal to the number of linear inequalities.

Scaling for linear constraints is applied *after* any continuous variable scaling.

For example, for variable scaling on continuous design variables x :

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear inequality constraints

$$\begin{aligned} a_L &\leq A_i x \leq a_U \\ a_L &\leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \\ a_L - A_i x_O &\leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \\ \tilde{a}_L &\leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \end{aligned}$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to [0,1].

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.4.37 linear_inequality_scales

- [Keywords Area](#)
- [variables](#)
- [linear_inequality_scales](#)

Characteristic values to scale linear inequalities

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

Description

Each real value in `linear_inequality_scales` is a nonzero characteristic value to be used in scaling each constraint. They only have effect when the associated method specifies `scaling`.

This keyword is required for `linear_inequality_scale_types` of 'value' and optional for 'auto'.

If a single real value is specified it will apply to all linear inequality constraints. Otherwise the number of values should be equal to the number of linear inequalities.

Scaling for linear constraints is applied *after* any continuous variable scaling.

For example, for variable scaling on continuous design variables x :

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear inequality constraints

$$\begin{aligned} a_L &\leq A_i x \leq a_U \\ a_L &\leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U \\ a_L - A_i x_O &\leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O \\ \tilde{a}_L &\leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U \end{aligned}$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.4.38 linear_equality_constraint_matrix

- [Keywords Area](#)
- [variables](#)
- [linear_equality_constraint_matrix](#)

Define coefficients of the linear equalities

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): REALLIST

Default: no linear equality constraints

Description

In the equality case, the constraint matrix A provides coefficients for the variables on the left hand side of:

$$Ax = a_t$$

The [linear_constraints](#) topics page (linked above) outlines a few additional things to consider when using linear constraints.

Examples

An optimization involving three variables, x_1 , x_2 , and x_3 , is to be performed. These variables must satisfy a pair of linear equality constraints:

$$1.5 \cdot x_1 + 1.0 \cdot x_2 = 5.0$$

$$3.0 \cdot x_1 - 4.0 \cdot x_3 = 0.0$$

The pair of constraints can be written in matrix form as:

$$\begin{bmatrix} 1.5 & 1.0 & 0.0 \\ 3.0 & 0.0 & -4.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5.0 \\ 0.0 \end{bmatrix}$$

The coefficient matrix and right hand side are expressed to Dakota in the [variables](#) section of the input file:

```
variables
  continuous_design 2
  descriptors 'x1' 'x2'

  linear_equality_constraint_matrix = 1.5  1.0  0.0
                                     3.0  0.0 -4.0

  linear_equality_targets = 5.0
                           0.0
```

For related examples, see the [linear_inequality_constraint_matrix](#) keyword page.

7.4.39 linear_equality_targets

- [Keywords Area](#)
- [variables](#)
- [linear_equality_targets](#)

Define target values for the linear equality constraints

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 0 .

Description

In the equality case, the targets a_t provide the equality constraint right hand sides:

$$Ax = a_t$$

If this is not specified, the defaults for the equality constraint targets enforce a value of 0. for each constraint:
 $Ax = 0.0$

Examples

Examples of specifying linear equality constraints to Dakota are provided on the [linear_equality_constraint_matrix](#) page.

7.4.40 linear_equality_scale_types

- [Keywords Area](#)
- [variables](#)
- [linear_equality_scale_types](#)

How to scale each linear equality constraint

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = "none"

Description

Each string in `linear_equality_scale_types` indicates the scaling type for each linear equality constraint. They only have effect when the associated method specifies `scaling`.

The options are:

- 'value' - characteristic value. If this is chosen, then [linear_equality_scales](#) must be specified; 'value' is assumed if scales are given without a `scale_types`

- 'auto' - automatic scaling.

If a single string is specified it will apply to all linear equality constraints. Otherwise the number of strings specified should be equal to the number of linear equalities.

Scaling for linear constraints is applied *after* any continuous variable scaling.

For example, for variable scaling on continuous design variables x :

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear equality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0,1]$.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.4.41 linear_equality_scales

- [Keywords Area](#)
- [variables](#)
- [linear_equality_scales](#)

Characteristic values to scale linear equalities

Topics

This keyword is related to the topics:

- [linear_constraints](#)

Specification

Alias: none

Argument(s): REALLIST

Default: vector values = 1 . (no scaling)

Description

Each real value in `linear_equality_scales` is a nonzero characteristic value to be used in scaling each constraint. They only have effect when the associated method specifies `scaling`.

This keyword is required for [linear_equality_scale_types](#) of 'value' and optional for 'auto'.

If a single real value is specified it will apply to all linear equality constraints. Otherwise the number of values should be equal to the number of linear equalities.

Scaling for linear constraints is applied *after* any continuous variable scaling.

For example, for variable scaling on continuous design variables x :

$$\tilde{x}^j = \frac{x^j - x_O^j}{x_M^j}$$

we have the following system for linear inequality constraints

$$a_L \leq A_i x \leq a_U$$

$$a_L \leq A_i (\text{diag}(x_M)\tilde{x} + x_O) \leq a_U$$

$$a_L - A_i x_O \leq A_i \text{diag}(x_M)\tilde{x} \leq a_U - A_i x_O$$

$$\tilde{a}_L \leq \tilde{A}_i \tilde{x} \leq \tilde{a}_U$$

and user-specified or automatically computed scaling multipliers are applied to this final transformed system, which accounts for continuous design variable scaling. When automatic scaling is in use for linear constraints they are linearly scaled by a computed characteristic value, but not affinely to $[0, 1]$.

See the scaling information under specific methods, e.g., `method=*-scaling` for details on how to use this keyword.

7.5 interface

- [Keywords Area](#)
- [interface](#)

Specifies how function evaluations will be performed in order to map the variables into the responses.

Topics

This keyword is related to the topics:

- [block](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		id_interface	Name the interface block; helpful when there are multiple

	Optional		analysis_drivers	Define how Dakota should run a function evaluation
	Optional		algebraic_-mappings	Use AMPL to define algebraic input-output mappings
	Optional		failure_capture	Determine how Dakota responds to analysis driver failure
	Optional		deactivate	Deactivate Dakota interface features for simplicity or efficiency
	Optional (<i>Choose One</i>)	Group 1	batch	Perform evaluations in batches
			asynchronous	Specify local evaluation or analysis concurrency
	Optional		evaluation_servers	Specify the number of evaluation servers when Dakota is run in parallel
	Optional		evaluation_-scheduling	Specify the scheduling of concurrent evaluations when Dakota is run in parallel
	Optional		processors_per_-evaluation	Specify the number of processors per evaluation server when Dakota is run in parallel
	Optional		analysis_servers	Specify the number of analysis servers when Dakota is run in parallel

	Optional	analysis_ - scheduling	Specify the scheduling of concurrent analyses when Dakota is run in parallel
--	-----------------	--	--

Description

The interface section in a Dakota input file specifies how function evaluations will be performed in order to map the variables into the responses. The term "interface" refers to the bridge between Dakota and the underlying simulation code.

In this context, a "function evaluation" is the series of operations that takes the variables and computes the responses. This can be comprised of one or many codes, scripts, and glue, which are generically referred to as "analysis drivers" (and optional input/output filters). The mapping actions of [analysis_drivers](#) may be combined with explicit [algebraic_mappings](#)

Parallelism Options

- The [asynchronous](#) keyword enables concurrent local function evaluations or analyses via operating system process management. Its child keywords allow tailoring the evaluation and analysis concurrency.
- The evaluation servers, scheduling mode (master, peer static or dynamic), and processor keywords allow a user to override Dakota's default evaluation configuration when running in parallel (MPI) mode.
- The analysis servers and scheduling mode (master, peer static or dynamic) keywords allow a user to override Dakota's default analysis configuration when running in parallel (MPI) mode.

Note: see [direct](#) for the specific `processors_per_analysis` specification supported for direct interfaces.

The ParallelLibrary class and the Parallel Computing chapter of the Users Manual [5] provide additional details on parallel configurations.

Theory

Function evaluations are performed using either interfaces to simulation codes, algebraic mappings, or a combination of the two.

When employing mappings with simulation codes, the interface invokes the simulation using either forks, direct function invocations, or computational grid invocations.

- In the fork case, Dakota will treat the simulation as a black-box and communication between Dakota and the simulation occurs through parameter and result files. This is the most common case.
- In the direct function case, the simulation is internal to Dakota and communication occurs through the function parameter list. The direct case can involve linked simulation codes or test functions which are compiled into the Dakota executable. The test functions allow for rapid testing of algorithms without process creation overhead or engineering simulation expense.
- The grid case is deprecated, but was an experiment in interfacing Dakota to distributed computing engines.

When employing algebraic mappings, the AMPL solver library [29] is used to evaluate a directed acyclic graph (D-AG) specification from a separate `stub.nl` file. Separate `stub.col` and `stub.row` files are also required to declare the string identifiers of the subset of inputs and outputs, respectively, that will be used in the algebraic mappings.

7.5.1 id_interface

- [Keywords Area](#)
- [interface](#)
- [id_interface](#)

Name the interface block; helpful when there are multiple

Topics

This keyword is related to the topics:

- [block_identifier](#)

Specification

Alias: none

Argument(s): STRING

Default: use of last interface parsed

Description

The optional `id_interface` keyword accepts a string that uniquely identifies this interface block. A model can then use this interface by specifying the same string in its `interface_pointer` specification.

Default Behavior

If the `id_interface` specification is omitted, a particular interface specification will be used by a model only if that model does not include an `interface_pointer` and the interface block was the last (or only) one parsed.

Usage Tips

- It is a best practice to always use explicit interface IDs and pointers to avoid confusion.
- If only one interface block exists, then `id_interface` can be safely omitted from the interface block (and `interface_pointer` omitted from the model specification(s)), since there is no ambiguity.

Examples

For example, a model specification including

```
model
  interface_pointer = 'I1'
```

will link to an interface with

```
id_interface = 'I1'
```

7.5.2 analysis_drivers

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)

Define how Dakota should run a function evaluation

Specification

Alias: none

Argument(s): STRINGLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		input_filter	Run a pre-processing script before the analysis drivers
	Optional		output_filter	Run a post-processing script after the analysis drivers
	Required (<i>Choose One</i>)	Interface Type (Group 1)	system	(Not recommended) Launch analysis drivers with a system call
			fork	Launch analysis drivers using fork command
			direct	Run analysis drivers that are linked-to or compiled-with Dakota
			plugin	Dynamically load a plugin analysis driver
			matlab	Run Matlab through a direct interface - requires special Dakota build
			python	Run Python through a Pybind11-based direct interface - requires a special Dakota build

			legacy_python	Run Python through a deprecated C-based direct interface - requires a special Dakota build
			scilab	Run Scilab through a direct interface - requires special Dakota build
			grid	Deprecated grid computing interface
	Optional		analysis_components	Provide additional identifiers to analysis drivers.

Description

The `analysis_drivers` keyword provides the names of one or more executable analysis programs or scripts, a.k.a. "drivers" which comprise a function evaluation. The optional and required sub-keywords specify how Dakota will manage directories and files, and run the driver(s).

Types of Interfaces

Dakota has two recommended ways of running analysis drivers:

- as an external processes (`fork`), or
- using internal code to couple to the analysis driver (`direct`)

Other options are available for advanced users, and are not as well documented, supported, or tested:

- external processes (`system`)
- internal coupling (`python`, `matlab`, `scilab`, `grid`)

Use Cases

The internally coupled codes have few options because many of the details are already handled with the coupling. Their behavior is described in the [direct](#) keyword.

For external processes using the [fork](#) keyword,

A function evaluation may comprise:

1. *A single analysis driver*: Function evaluation, including all pre- and post-processing is contained entirely within a single script/executable.
2. *A single analysis driver with filters*: Function evaluation is explicitly split into pre-processing (performed by the input filter), analysis, and post-processing (by the output filter).
3. *A single analysis driver with environment variables*: Function evaluation is contained within one analysis driver, but it requires environment variables to be set before running.
4. *Multiple analysis drivers*: Drivers are run sequentially or concurrently (See the `asynchronous` keyword) and can have any of the above options as well.

For fork and system interfaces, the `analysis_driver` list contains the names of one or more executable programs or scripts taking parameters files as input and producing results files as output. The first field in each analysis driver string must be an executable program or script for Dakota to spawn to perform the function evaluation. Drivers support:

- One set of nested quotes, for arguments with spaces
- Dakota will define special environment variables `DAKOTA_PARAMETERS_FILE` and `DAKOTA_RESULTS_FILE` which can be used in the driver script.
- Dakota will replace the tokens `{PARAMETERS}` and `{RESULTS}` in an analysis driver string with the names of the parameters and results files for that analysis/evaluation. Along with the [verbatim](#) keyword, which prevents Dakota from appending the names of the parameters and results files as command line arguments, this feature provides users with greater control over how their analysis drivers are invoked by Dakota.
- Variable definitions preceding the executable program or script, such as `'MY_VAR=2 run_analysis.sh'` are no longer supported.

For details and examples see the Simulation Interface Components section of the Interfaces chapter of the User's Manual; for details on the filters and environment variables, see the subsection on Syntax for Filter and Driver Strings.

Examples

Examples:

1. `analysis_drivers = 'run_simulation_part1.sh' 'run_simulation_part2.sh'`
2. `analysis_driver = 'run_simulation.sh -option "option 1"'`
3. `analysis_driver = 'simulation.exe -option value -dakota_params $DAKOTA_PARAMETERS_FILE -input sim.in -dakota_results $DAKOTA_RESULTS_FILE'`

FAQ

Where will Dakota look for the analysis_driver? Dakota will locate `analysis_driver` programs first in (or relative to) the present working directory (`."`, the `interface-analysis_drivers-fork-work_directory` if used, otherwise the directory in which Dakota is started), then the directory from which Dakota is started, then using the system `$PATH` environment variable (`Path%` on Windows).

Where should the driver be located? When the driver is a script it is most commonly placed in the same directory as the Dakota input file. When using a [work_directory](#), Dakota will also look for drivers in the specified working directory, so `link_files` or `copy_files` may specify the driver to get copied or linked into the work directory. When executable programs are used as drivers, they are often elsewhere on the filesystem. These can be specified using absolute paths, or by prepending the `PATH` environment variable so Dakota finds them.

What if Dakota fails to run my analysis_driver? Prepend the absolute location of the driver to the `PATH` environment variable before running Dakota, or specify an absolute path to the driver in the Dakota input file.

7.5.2.1 input_filter

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [input_filter](#)

Run a pre-processing script before the analysis drivers

Specification

Alias: none

Argument(s): STRING

Default: no input filter

Description

The optional `input_filter` and `output_filter` specifications provide the names of separate pre- and post-processing programs or scripts which assist in mapping Dakota parameters files into analysis input files and mapping analysis output files into Dakota results files, respectively.

If there is only a single analysis driver, then it is usually most convenient to combine pre- and post-processing requirements into a single analysis driver script and omit the separate input and output filters. However, in the case of multiple analysis drivers, the input and output filters provide a convenient location for non-repeated pre- and post-processing requirements. That is, input and output filters are only executed once per function evaluation, regardless of the number of analysis drivers, which makes them convenient locations for data processing operations that are shared among the analysis drivers.

The `verbatim` keyword applies to input and output filters as well as analysis drivers, and Dakota also will substitute the names of the parameters and results files for the tokens `{PARAMETERS}` and `{RESULTS}` in input and output filter strings, as explained in the documentation for [analysis_drivers](#).

7.5.2.2 `output_filter`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [output_filter](#)

Run a post-processing script after the analysis drivers

Specification

Alias: none

Argument(s): STRING

Default: no output filter

Description

The optional `input_filter` and `output_filter` specifications provide the names of separate pre- and post-processing programs or scripts which assist in mapping Dakota parameters files into analysis input files and mapping analysis output files into Dakota results files, respectively.

If there is only a single analysis driver, then it is usually most convenient to combine pre- and post-processing requirements into a single analysis driver script and omit the separate input and output filters. However, in the case of multiple analysis drivers, the input and output filters provide a convenient location for non-repeated pre- and post-processing requirements. That is, input and output filters are only executed once per function evaluation, regardless of the number of analysis drivers, which makes them convenient locations for data processing operations that are shared among the analysis drivers.

The `verbatim` keyword applies to input and output filters as well as analysis drivers, and Dakota also will substitute the names of the parameters and results files for the tokens `{PARAMETERS}` and `{RESULTS}` in input and output filter strings, as explained in the documentation for [analysis_drivers](#).

7.5.2.3 system

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)

(Not recommended) Launch analysis drivers with a system call

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		parameters_file	Specify the name of the parameters file
	Optional		results_file	Specify the name of the results file
	Optional		file_tag	Tag each parameters & results file name with the function evaluation number
	Optional		file_save	Keep the parameters & results files after the analysis driver completes
	Optional		labeled	Requires correct function value labels in results file
	Optional		aprepro	Write parameters files in APREPRO syntax
	Optional		work_directory	Perform each function evaluation in a separate working directory
	Optional		allow_existing_results	Change how Dakota deals with existing results files

	Optional	verbatim	Specify the command Dakota uses to launch analysis driver(s) and filters
--	-----------------	--------------------------	--

Description

The system call interface is included in Dakota for portability and backward compatibility. Users are strongly encouraged to use the `fork` interface if possible, reverting to system only when necessary. To enable the system call interface, replace the `fork` keyword with `system`. All other keywords have identical meanings to those for the `fork` interface

See Also

These keywords may also be of interest:

- [fork](#)

7.5.2.3.1 `parameters_file`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [parameters_file](#)

Specify the name of the parameters file

Specification

Alias: none

Argument(s): STRING

Default: Unix temp files

Description

The parameters file is used by Dakota to pass the parameter values to the analysis driver. The name of the file can be optionally specified using the `parameters_file` keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., `/tmp/dakota_params_aaaa0886`).

7.5.2.3.2 `results_file`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)

- [results_file](#)

Specify the name of the results file

Specification

Alias: none

Argument(s): STRING

Default: Unix temp files

Description

The results file must be written by the analysis driver. It is read by Dakota to determine the response values for each function evaluation.

The name of the file can be optionally specified using the `results_file` keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., `/tmp/dakota_results_aaaa0886`).

7.5.2.3.3 file_tag

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [file_tag](#)

Tag each parameters & results file name with the function evaluation number

Specification

Alias: none

Argument(s): none

Default: no tagging

Description

If this keyword is used, Dakota will append a period and the function evaluation number to the names of the parameter and results files.

Default Behavior If this keyword is omitted, the default is no file tagging.

Usage Tips

- File tagging is most useful when multiple function evaluations are running simultaneously using files in a shared disk space. The analysis driver will be able to infer the function evaluation number from the file names.
- Note that when the `file_save` keyword is used, Dakota automatically renames parameters and results files, giving them tags after execution of the analysis driver if they otherwise would be overwritten by the next evaluation.

Examples

If the following is included in the `interface` section of the Dakota input:

```
parameters_file = params.in
results_file = results.out
file_tag
```

Then for the 3rd evaluation, Dakota will write `params.in.3`, and will expect `results.out.3` to be written by the analysis driver.

7.5.2.3.4 file_save

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [file_save](#)

Keep the parameters & results files after the analysis driver completes

Specification

Alias: none

Argument(s): none

Default: file cleanup

Description

If `file_save` is used, Dakota will not delete the parameters and results files after the function evaluation is completed.

The default behavior is NOT to save these files.

If `file_tag` is not specified and the saved files would be overwritten by a future evaluation, Dakota renames them after the analysis driver has run by tagging them with the evaluation number.

File saving is most useful when debugging the data communication between Dakota and the simulation.

7.5.2.3.5 labeled

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [labeled](#)

Requires correct function value labels in results file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: Function value labels optional

Description

The `labeled` keyword directs Dakota to enforce a stricter results file format and enables more detailed error reporting.

When the `labeled` keyword is used, function values in results files must be accompanied by their corresponding descriptors. If the user did not supply response [descriptors](#) in her Dakota input file, then Dakota auto-generated descriptors are expected.

Distinct error messages are emitted for function values that are out-of-order, repeated, or missing. Labels that appear without a function value and unexpected data are also reported as errors. Dakota attempts to report all errors in a results file, not just the first it encounters. After reporting results file errors, Dakota aborts.

Labels for analytic gradients and Hessians currently are not supported.

Although the `labeled` keyword is optional, its use is recommended to help catch and identify problems with results files. The User's Manual contains further information about the results file format.

Default Behavior

By default, Dakota does not require labels for function values, and ignores them if they are present.

7.5.2.3.6 `aprepro`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [aprepro](#)

Write parameters files in APREPRO syntax

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: `dprepro`

Argument(s): none

Default: standard parameters file format

Description

The format of data in the parameters files can be modified for direct usage with the APREPRO pre-processing tool [78] using the `aprepro` specification

Without this keyword, the parameters file are written in DPrePro format. DPrePro is a utility included with Dakota, described in the Users Manual [5].

7.5.2.3.7 `work_directory`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [work_directory](#)

Perform each function evaluation in a separate working directory

Specification

Alias: none

Argument(s): none

Default: no work directory

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		named	The base name of the work directory created by Dakota
	Optional		directory_tag	Tag each work directory with the function evaluation number
	Optional		directory_save	Preserve the work directory after function evaluation completion
	Optional		link_files	Paths to be linked into each working directory
	Optional		copy_files	Files and directories to be copied into each working directory

	Optional	replace	Overwrite existing files within a work directory
--	-----------------	-------------------------	--

Description

When performing concurrent evaluations, it is typically necessary to cloister simulation input and output files in separate directories to avoid conflicts. When the `work_directory` feature is enabled, Dakota will create a directory for each evaluation, with optional tagging (`directory_tag`) and saving (`directory_save`), as with files, and execute the analysis driver from that working directory.

The directory may be named with a string, or left anonymous to use an automatically-generated directory in the system's temporary file space, e.g., `/tmp/dakota_work_c93vb71z/`. The optional `link_files` and `copy_files` keywords specify files or directories which should appear in each working directory.

When using `work_directory`, the [analysis_drivers](#) may be given by an absolute path, located in (or relative to) the startup directory alongside the Dakota input file, in the list of template files linked or copied, or on the \$PATH (Path% on Windows).

7.5.2.3.7.1 named

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [work_directory](#)
- [named](#)

The base name of the work directory created by Dakota

Specification

Alias: none

Argument(s): STRING

Default: `dakota_work_XXXXXXXX`

Description

The `named` keyword is followed by a string, indicating the name of the work directory created by Dakota. If relative, the work directory will be created relative to the directory from which Dakota is invoked.

If `named` is not used, the default work directory is a temporary directory with a system-generated name (e.g., `/tmp/dakota_work_c93vb71z/`).

See Also

These keywords may also be of interest:

- [directory_tag](#)
- [directory_save](#)

7.5.2.3.7.2 `directory_tag`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [work_directory](#)
- [directory_tag](#)

Tag each work directory with the function evaluation number

Specification

Alias: `dir_tag`

Argument(s): none

Default: no work directory tagging

Description

If this keyword is used, Dakota will append a period and the function evaluation number to the work directory names.

If this keyword is omitted, the default is no tagging, and the same work directory will be used for ALL function evaluations. Tagging is most useful when multiple function evaluations are running simultaneously.

7.5.2.3.7.3 `directory_save`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [work_directory](#)
- [directory_save](#)

Preserve the work directory after function evaluation completion

Specification

Alias: `dir_save`

Argument(s): none

Default: remove work directory

Description

By default, when a working directory is created by Dakota using the `work_directory` keyword, it is deleted after the evaluation is completed. The `directory_save` keyword will cause Dakota to leave (not delete) the directory.

7.5.2.3.7.4 link_files

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [work_directory](#)
- [link_files](#)

Paths to be linked into each working directory

Specification

Alias: none

Argument(s): STRINGLIST

Default: no linked files

Description

Specifies the paths (files or directories) that will be symbolically linked from each working directory. Wildcards using * and ? are permitted. Linking is space-saving and useful for files not modified during the function evaluation. However, not all filesystems support linking, for example, support on Windows varies.

Examples

Specifying

```
link_files = 'siminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```
workdir/siminput*.in # links to each of rundir / siminput*.in
workdir/simdir1/    # whole directory simdir1 linked
workdir/*          # each entry in directory simdir2 linked
```

7.5.2.3.7.5 copy_files

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [work_directory](#)
- [copy_files](#)

Files and directories to be copied into each working directory

Specification

Alias: none

Argument(s): STRINGLIST

Default: no copied files

Description

Specifies the files or directories that will be recursively copied into each working directory. Wildcards using * and ? are permitted.

Examples

Specifying

```
copy_files = 'siminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```
workdir/siminput*.in # files rundir/siminput*.in copied
workdir/simdir1/     # whole directory simdir1 recursively copied
workdir/*            # contents of directory simdir2 recursively copied
```

where rundir is the directory in which Dakota was started.

7.5.2.3.7.6 replace

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [work_directory](#)
- [replace](#)

Overwrite existing files within a work directory

Specification

Alias: none

Argument(s): none

Default: do not overwrite files

Description

By default, Dakota will not overwrite any existing files in a work directory. The `replace` keyword changes this behavior to force overwriting.

7.5.2.3.8 allow_existing_results

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [allow_existing_results](#)

Change how Dakota deals with existing results files

Specification

Alias: none

Argument(s): none

Default: results files removed before each evaluation

Description

By default Dakota will remove existing results files before invoking the `analysis_driver` to avoid problems created by stale files in the current directory. To override this behavior and not delete existing files, specify `allow_existing_results`.

7.5.2.3.9 verbatim

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [system](#)
- [verbatim](#)

Specify the command Dakota uses to launch analysis driver(s) and filters

Specification

Alias: none

Argument(s): none

Default: driver/filter invocation syntax augmented with file names

Description

The typical commands that Dakota uses to launch analysis drivers are:

```
> analysis_driver parameters_file_name results_file_name
```

Dakota will automatically arrange the executables and file names.

If the analysis driver requires a different syntax, the entire command can be specified as the analysis driver and the `verbatim` keyword will tell Dakota to use this as the command.

Note, this will not allow the use of `file_tag`, because the exact command must be specified.

For additional information on invocation syntax, see the Interfaces chapter of the Users Manual [5].

Examples

In the following example, the `analysis_driver` command is run without any edits from Dakota.

```
interface
  analysis_driver = "matlab -nodesktop -nojvm -r 'MatlabDriver_hardcoded_filenames; exit' "
  fork
    parameters_file 'params.in'
    results_file 'results.out'
    verbatim # this tells Dakota to fork the command exactly as written, instead of appending I/O filenames
```

The `-r` flag identifies the commands that will be run by matlab. The Matlab script has the `parameters_file` and `results_file` names hardcoded, so no additional arguments are required.

7.5.2.4 fork

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)

Launch analysis drivers using fork command

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		parameters_file	Specify the name of the parameters file
	Optional		results_file	Specify the name of the results file
	Optional		file_tag	Tag each parameters & results file name with the function evaluation number

	Optional	file_save	Keep the parameters & results files after the analysis driver completes
	Optional	labeled	Requires correct function value labels in results file
	Optional	aprepro	Write parameters files in APREPRO syntax
	Optional	work_directory	Perform each function evaluation in a separate working directory
	Optional	allow_existing_results	Change how Dakota deals with existing results files
	Optional	verbatim	Specify the command Dakota uses to launch analysis driver(s) and filters

Description

The `fork` interface is the most common means by which Dakota launches a separate application analysis process.

The `fork` interface is recommended over `system` for most analysis drivers that are external to Dakota, i.e., any driver not linked in via the `direct` interface.

As explained in the Users Manual, the parameters and results file names are passed on the command line to the analysis driver(s). If input/output filters are specified, they will be run before/after the analysis drivers. The `verbatim` keyword is used to modify the default driver/filter commands.

For additional information on invocation syntax, see the Interfaces chapter of the Users Manual [5].

Examples

Spawn (fork) an external executable/script called 'rosenbrock' which reads variables from `params.in` and writes responses to `results.out`. Preserve the analysis files for each function evaluation with tag and save.

```
interface
  analysis_drivers = 'rosenbrock'
  fork
    parameters_file = 'params.in'
    results_file    = 'results.out'
    file_tag
    file_save
```

7.5.2.4.1 parameters_file

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)

- [fork](#)
- [parameters_file](#)

Specify the name of the parameters file

Specification

Alias: none

Argument(s): STRING

Default: Unix temp files

Description

The parameters file is used by Dakota to pass the parameter values to the analysis driver. The name of the file can be optionally specified using the `parameters_file` keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., `/tmp/dakota_params_aaaa0886`).

7.5.2.4.2 results_file

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [results_file](#)

Specify the name of the results file

Specification

Alias: none

Argument(s): STRING

Default: Unix temp files

Description

The results file must be written by the analysis driver. It is read by Dakota to determine the response values for each function evaluation.

The name of the file can be optionally specified using the `results_file` keyword.

If this is not specified, the default data transfer files are temporary files with system-generated names (e.g., `/tmp/dakota_results_aaaa0886`).

7.5.2.4.3 file_tag

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)

- [fork](#)
- [file_tag](#)

Tag each parameters & results file name with the function evaluation number

Specification

Alias: none

Argument(s): none

Default: no tagging

Description

If this keyword is used, Dakota will append a period and the function evaluation number to the names of the parameter and results files.

Default Behavior If this keyword is omitted, the default is no file tagging.

Usage Tips

- File tagging is most useful when multiple function evaluations are running simultaneously using files in a shared disk space. The analysis driver will be able to infer the function evaluation number from the file names.
- Note that when the `file_save` keyword is used, Dakota automatically renames parameters and results files, giving them tags after execution of the analysis driver if they otherwise would be overwritten by the next evaluation.

Examples

If the following is included in the `interface` section of the Dakota input:

```
parameters_file = params.in
results_file = results.out
file_tag
```

Then for the 3rd evaluation, Dakota will write `params.in.3`, and will expect `results.out.3` to be written by the analysis driver.

7.5.2.4.4 `file_save`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [file_save](#)

Keep the parameters & results files after the analysis driver completes

Specification

Alias: none

Argument(s): none

Default: file cleanup

Description

If `file_save` is used, Dakota will not delete the parameters and results files after the function evaluation is completed.

The default behavior is NOT to save these files.

If `file_tag` is not specified and the saved files would be overwritten by a future evaluation, Dakota renames them after the analysis driver has run by tagging them with the evaluation number.

File saving is most useful when debugging the data communication between Dakota and the simulation.

7.5.2.4.5 labeled

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [labeled](#)

Requires correct function value labels in results file

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: Function value labels optional

Description

The `labeled` keyword directs Dakota to enforce a stricter results file format and enables more detailed error reporting.

When the `labeled` keyword is used, function values in results files must be accompanied by their corresponding descriptors. If the user did not supply response [descriptors](#) in her Dakota input file, then Dakota auto-generated descriptors are expected.

Distinct error messages are emitted for function values that are out-of-order, repeated, or missing. Labels that appear without a function value and unexpected data are also reported as errors. Dakota attempts to report all errors in a results file, not just the first it encounters. After reporting results file errors, Dakota aborts.

Labels for analytic gradients and Hessians currently are not supported.

Although the `labeled` keyword is optional, its use is recommended to help catch and identify problems with results files. The User's Manual contains further information about the results file format.

Default Behavior

By default, Dakota does not require labels for function values, and ignores them if they are present.

7.5.2.4.6 `aprepro`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [aprepro](#)

Write parameters files in APREPRO syntax

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: `dprepro`

Argument(s): none

Default: standard parameters file format

Description

The format of data in the parameters files can be modified for direct usage with the APREPRO pre-processing tool [78] using the `aprepro` specification

Without this keyword, the parameters file are written in DPrePro format. DPrePro is a utility included with Dakota, described in the Users Manual [5].

7.5.2.4.7 `work_directory`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [work_directory](#)

Perform each function evaluation in a separate working directory

Specification

Alias: none

Argument(s): none

Default: no work directory

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		named	The base name of the work directory created by Dakota
	Optional		directory_tag	Tag each work directory with the function evaluation number
	Optional		directory_save	Preserve the work directory after function evaluation completion
	Optional		link_files	Paths to be linked into each working directory
	Optional		copy_files	Files and directories to be copied into each working directory
	Optional		replace	Overwrite existing files within a work directory

Description

When performing concurrent evaluations, it is typically necessary to cloister simulation input and output files in separate directories to avoid conflicts. When the `work_directory` feature is enabled, Dakota will create a directory for each evaluation, with optional tagging (`directory_tag`) and saving (`directory_save`), as with files, and execute the analysis driver from that working directory.

The directory may be `named` with a string, or left anonymous to use an automatically-generated directory in the system's temporary file space, e.g., `/tmp/dakota_work_c93vb71z/`. The optional `link_files` and `copy_files` keywords specify files or directories which should appear in each working directory.

When using `work_directory`, the `analysis_drivers` may be given by an absolute path, located in (or relative to) the startup directory alongside the Dakota input file, in the list of template files linked or copied, or on the `$PATH` (Path% on Windows).

7.5.2.4.7.1 `named`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)

- [fork](#)
- [work_directory](#)
- [named](#)

The base name of the work directory created by Dakota

Specification

Alias: none

Argument(s): STRING

Default: dakota_work_XXXXXXXX

Description

The `named` keyword is followed by a string, indicating the name of the work directory created by Dakota. If relative, the work directory will be created relative to the directory from which Dakota is invoked.

If `named` is not used, the default work directory is a temporary directory with a system-generated name (e.g., `/tmp/dakota_work_c93vb71z/`).

See Also

These keywords may also be of interest:

- [directory_tag](#)
- [directory_save](#)

7.5.2.4.7.2 [directory_tag](#)

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [work_directory](#)
- [directory_tag](#)

Tag each work directory with the function evaluation number

Specification

Alias: `dir_tag`

Argument(s): none

Default: no work directory tagging

Description

If this keyword is used, Dakota will append a period and the function evaluation number to the work directory names.

If this keyword is omitted, the default is no tagging, and the same work directory will be used for ALL function evaluations. Tagging is most useful when multiple function evaluations are running simultaneously.

7.5.2.4.7.3 `directory_save`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [work_directory](#)
- [directory_save](#)

Preserve the work directory after function evaluation completion

Specification

Alias: `dir_save`

Argument(s): none

Default: remove work directory

Description

By default, when a working directory is created by Dakota using the `work_directory` keyword, it is deleted after the evaluation is completed. The `directory_save` keyword will cause Dakota to leave (not delete) the directory.

7.5.2.4.7.4 `link_files`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [work_directory](#)
- [link_files](#)

Paths to be linked into each working directory

Specification

Alias: none

Argument(s): STRINGLIST

Default: no linked files

Description

Specifies the paths (files or directories) that will be symbolically linked from each working directory. Wildcards using * and ? are permitted. Linking is space-saving and useful for files not modified during the function evaluation. However, not all filesystems support linking, for example, support on Windows varies.

Examples

Specifying

```
link_files = 'siminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```
workdir/siminput*.in # links to each of rundir / siminput*.in
workdir/simdir1/     # whole directory simdir1 linked
workdir/*            # each entry in directory simdir2 linked
```

7.5.2.4.7.5 copy_files

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [work_directory](#)
- [copy_files](#)

Files and directories to be copied into each working directory

Specification

Alias: none

Argument(s): STRINGLIST

Default: no copied files

Description

Specifies the files or directories that will be recursively copied into each working directory. Wildcards using * and ? are permitted.

Examples

Specifying

```
copy_files = 'siminput*.in' '/path/to/simdir1' 'simdir2/*'
```

will create copies

```
workdir/siminput*.in # files rundir/siminput*.in copied
workdir/simdir1/     # whole directory simdir1 recursively copied
workdir/*            # contents of directory simdir2 recursively copied
```

where rundir is the directory in which Dakota was started.

7.5.2.4.7.6 replace

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [work_directory](#)
- [replace](#)

Overwrite existing files within a work directory

Specification

Alias: none

Argument(s): none

Default: do not overwrite files

Description

By default, Dakota will not overwrite any existing files in a work directory. The `replace` keyword changes this behavior to force overwriting.

7.5.2.4.8 allow_existing_results

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [allow_existing_results](#)

Change how Dakota deals with existing results files

Specification

Alias: none

Argument(s): none

Default: results files removed before each evaluation

Description

By default Dakota will remove existing results files before invoking the `analysis_driver` to avoid problems created by stale files in the current directory. To override this behavior and not delete existing files, specify `allow_existing_results`.

7.5.2.4.9 verbatim

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [fork](#)
- [verbatim](#)

Specify the command Dakota uses to launch analysis driver(s) and filters

Specification

Alias: none

Argument(s): none

Default: driver/filter invocation syntax augmented with file names

Description

The typical commands that Dakota uses to launch analysis drivers are:

```
> analysis_driver parameters_file_name results_file_name
```

Dakota will automatically arrange the executables and file names.

If the analysis driver requires a different syntax, the entire command can be specified as the analysis driver and the `verbatim` keyword will tell Dakota to use this as the command.

Note, this will not allow the use of `file_tag`, because the exact command must be specified.

For additional information on invocation syntax, see the Interfaces chapter of the Users Manual [5].

Examples

In the following example, the `analysis_driver` command is run without any edits from Dakota.

```
interface
  analysis_driver = "matlab -nodesktop -nojvm -r 'MatlabDriver_hardcoded_filenames; exit' "
  fork
    parameters_file 'params.in'
    results_file 'results.out'
    verbatim # this tells Dakota to fork the command exactly as written, instead of appending I/O filenames
```

The `-r` flag identifies the commands that will be run by matlab. The Matlab script has the `parameters_file` and `results_file` names hardcoded, so no additional arguments are required.

7.5.2.5 direct

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [direct](#)

Run analysis drivers that are linked-to or compiled-with Dakota

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		processors_per_analysis	Specify the number of processors per analysis when Dakota is run in parallel

Description

Direct interfaces are used to compile/link simulation programs into Dakota and to invoke Dakota's built-in algebraic test problems.

Direct simulation interfaces communicate variable and response data in-core instead of through the filesystem. This typically requires modification to simulator programs so that they can be linked into Dakota; however it can be more efficient due to elimination of external processes and auxilliary simulator output, more accurate due to higher numerics, and more flexible in terms of MPI parallelism.

Direct interfaces are also used to invoke internal test functions that perform parameter to response mappings for simple functions as inexpensively as possible. These problems are compiled directly into the Dakota executable as part of the direct function interface class and are used for algorithm testing.

Dakota supports direct interfaces to a few select simulation codes such as Matlab, Python, and Scilab. Another example is ModelCenter, a commercial simulation management framework from Phoenix Integration. To utilize this interface, a user must first define the simulation specifics within a ModelCenter session and then save these definitions to a ModelCenter configuration file. The `analysis_components` specification provides the means to communicate this configuration file to Dakota's ModelCenter interface.

Examples

The rosenbrock function is available as an executable, which can be launched with `fork`, and is also compiled with Dakota. The internal version can be used with:

```
interface
  analysis_drivers = 'rosenbrock'
  direct
```

7.5.2.5.1 processors_per_analysis

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [direct](#)
- [processors_per_analysis](#)

Specify the number of processors per analysis when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Default: automatic (see discussion)

Description

For direct function interfaces, `processors_per_analysis` can be used to specify multiprocessor analysis partitions. As with the `evaluation_servers`, `analysis_servers`, `evaluation_self_scheduling`, `evaluation_static_scheduling`, `analysis_self_scheduling`, and `analysis_static_scheduling` specifications, `processors_per_analysis` provides a means for the user to override the automatic parallel configuration (refer to `ParallelLibrary` and the Parallel Computing chapter of the Users Manual [5]) for the number of processors used for each analysis partition.

Usage Tips

- If both `analysis_servers` and `processors_per_analysis` are specified and they are not in agreement, then `analysis_servers` takes precedence.

7.5.2.6 plugin

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [plugin](#)

Dynamically load a plugin analysis driver

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		library_path	Path to the plugin shared object file

Description

Dakota 6.16 allows for run-time loading of a C++ or Python plugin analysis driver as a shared object file.

Python plugins support batch evaluations ([batch](#)) through a list of dictionaries.

Examples

See `dakota/src/plugins` and `dakota/share/dakota/test/dakota_plugin.in` for C++ and Python demos. These have only been tested on Linux.

7.5.2.6.1 `library_path`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [plugin](#)
- [library_path](#)

Path to the plugin shared object file

Specification

Alias: none

Argument(s): STRING

Default: none

Description

Dakota 6.16 requires the path to the plugin. We plan to make this specification more flexible in a future release.

7.5.2.7 `matlab`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [matlab](#)

Run Matlab through a direct interface - requires special Dakota build

Specification

Alias: none

Argument(s): none

Description

Dakota supports a library-linked interface to Matlab, but it must be explicitly enabled when compiling Dakota from source. Consult the Users Manual [5] for discussion and examples. Contact the Dakota users mailing list for assistance building and using Dakota with these interfaces.

The [analysis_drivers](#) specifies a Matlab file which implements the parameter to response mapping.

Examples

See `dakota/share/dakota/examples/users/MATLAB/linked`

7.5.2.8 python

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [python](#)

Run Python through a Pybind11-based direct interface - requires a special Dakota build

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		numpy	Enable the use of numpy in Dakota's Python interface

Description

Dakota supports a library-linked interface to Python, but it must be explicitly enabled when compiling Dakota from source. Consult the Users Manual [5] for discussion, examples, and documentation of the dictionary-based API. Contact the Dakota users mailing list for assistance building and using Dakota with these interfaces.

The [analysis_drivers](#) keyword specifies a Python module:function which implements the parameter to response mapping. List data structures are the default, but NumPy is also supported, if enabled in the build.

Batch evaluations ([batch](#)) are supported through a list of dictionaries.

Examples

See `dakota/share/dakota/examples/official/drivers/Python/linked` and `dakota/share/dakota/examples/Python/linked_di` for an example of its integration with `dakota.interfacing`.

7.5.2.8.1 numpy

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [python](#)
- [numpy](#)

Enable the use of numpy in Dakota's Python interface

Specification

Alias: none

Argument(s): none

Default: Python list dataflow

Description

When the `numpy` keyword is used, Dakota expects responses in the form of a Python dictionary of numpy arrays. See the example in `dakota/share/dakota/examples/official/drivers/Python/linked`.

7.5.2.9 legacy_python

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [legacy_python](#)

Run Python through a deprecated C-based direct interface - requires a special Dakota build

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>numpy</code>	Enable the use of numpy in Dakota's Python interface

Description

This interface has been superseded by a Pybind11 implementation [python](#) .

Dakota supports a library-linked interface to Python, but it must be explicitly enabled when compiling Dakota from source. The [analysis_drivers](#) keyword specifies a Python module:function which implements the parameter to response mapping. List data structures are the default, but NumPy is also supported, if enabled in the build.

7.5.2.9.1 numpy

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [legacy_python](#)
- [numpy](#)

Enable the use of numpy in Dakota's Python interface

Specification

Alias: none

Argument(s): none

Default: Python list dataflow

Description

When the `numpy` keyword is used, Dakota expects responses in the form of a Python dictionary of numpy arrays.

7.5.2.10 `scilab`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [scilab](#)

Run Scilab through a direct interface - requires special Dakota build

Specification

Alias: none

Argument(s): none

Description

Dakota supports a library-linked interface to Scilab, but it must be explicitly enabled when compiling Dakota from source. Consult the Users Manual [5] for discussion and examples. Contact the Dakota users mailing list for assistance building and using Dakota with these interfaces.

The [analysis_drivers](#) specifies a Scilab file which implements the parameter to response mapping.

Examples

See `dakota/share/dakota/examples/users/Scilab/linked`

7.5.2.11 `grid`

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [grid](#)

Deprecated grid computing interface

Specification

Alias: none

Argument(s): none

Description

Interface Dakota directly to a grid (distributed) computing engine. This deprecated capability was historically used for interfaces with IDEA and JAVASpaces in the past and was intended as a placeholder for future work with Condor and/or Globus services. It is not currently operational.

7.5.2.12 analysis_components

- [Keywords Area](#)
- [interface](#)
- [analysis_drivers](#)
- [analysis_components](#)

Provide additional identifiers to analysis drivers.

Specification

Alias: none

Argument(s): STRINGLIST

Default: no additional identifiers

Description

The optional `analysis_components` specification allows the user to provide additional identifiers (e.g., mesh file names) for use by the analysis drivers. This is particularly useful when the same analysis driver is to be reused multiple times for slightly different analyses. The specific content within the strings is open-ended and can involve whatever syntax is convenient for a particular analysis driver. The number of analysis components n_c should be an integer multiple of the number of drivers n_d , and the first n_c/n_d component strings will be passed to the first driver, etc.

7.5.3 algebraic_mappings

- [Keywords Area](#)
- [interface](#)
- [algebraic_mappings](#)

Use AMPL to define algebraic input-output mappings

Specification

Alias: none

Argument(s): STRING

Default: no algebraic mappings

Description

Dakota can evaluate algebraic input-output mappings using AMPL [26]. The mappings are expressed in 3 files: `stub.nl`, `stub.col`, and `stub.row`, where `stub` is a particular root name describing a particular problem. The file names are communicated to Dakota using the `algebraic_mappings` keyword. It may either specify the full `stub.nl` filename, or alternatively, just the `stub` basename.

Dakota then extracts the input and output identifier strings from `stub.col` and `stub.row` and employs the AMPL solver library [29] to process the directed acyclic graphc (DAG) specification in `stub.nl`. The variable and objective function names declared within AMPL should be a subset of the variable and response descriptors specified in the [variables](#) and [responses](#) blocks. Ordering is not important, as Dakota will reorder data as needed.

Examples

An interface employing both algebraic and simulation-based mappings. The results from the individual algebraic and simulation mappings are overlaid based on the variable and response descriptors used by the individual mappings.

```
interface,
  algebraic_mappings = 'ampl/fma.nl'
  fork
    analysis_driver = 'text_book'
    parameters_file = 'tb.in'
    results_file    = 'tb.out'
```

7.5.4 failure_capture

- [Keywords Area](#)
- [interface](#)
- [failure_capture](#)

Determine how Dakota responds to analysis driver failure

Specification

Alias: none

Argument(s): none

Default: abort

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Failure Mitigation (Group 1)	abort	(Default) Abort the Dakota job
			retry	Rerun failed analyses
			recover	Substitute dummy values for the responses

			continuation	Cause Dakota to step toward the failed "target" simulation from a nearby successful "source"
--	--	--	------------------------------	--

Description

Dakota can deal with analysis failure in a few ways.

The first step is that Dakota must detect analysis failure. Importantly, Dakota always expects a results file to be written by the analysis driver, even when a failure has occurred. If the file does not exist when the analysis driver exits, a Dakota error results, causing Dakota itself to terminate. The analysis driver communicates an analysis failure to Dakota by writing a results file beginning with the (case-insensitive) word "fail". Any file contents after "fail" are ignored.

Once Dakota detects analysis failure, the failure can be mitigated in four ways:

- [abort](#) (the default)
- [retry](#)
- [recover](#)
- [continuation](#)

Refer to the Simulation Code Failure Capturing chapter of the Users Manual [5] for additional information.

7.5.4.1 abort

- [Keywords Area](#)
- [interface](#)
- [failure_capture](#)
- [abort](#)

(Default) Abort the Dakota job

Specification

Alias: none

Argument(s): none

Description

Stop the Dakota job, as well as any other running analysis drivers when a failure is communicated.

7.5.4.2 retry

- [Keywords Area](#)
- [interface](#)
- [failure_capture](#)

- [retry](#)

Rerun failed analyses

Specification

Alias: none

Argument(s): INTEGER

Description

Number of times to rety a failing analysis

7.5.4.3 recover

- [Keywords Area](#)
- [interface](#)
- [failure_capture](#)
- [recover](#)

Substitute dummy values for the responses

Specification

Alias: none

Argument(s): REALLIST

Description

When a simulation failure is detected, substitute the provided dummy function values in the response. Gradient and Hessian are not supported.

7.5.4.4 continuation

- [Keywords Area](#)
- [interface](#)
- [failure_capture](#)
- [continuation](#)

Cause Dakota to step toward the failed "target" simulation from a nearby successful "source"

Specification

Alias: none

Argument(s): none

Description

When `failure_capture continuation` is enabled and an evaluation fails, then Dakota will attempt to march incrementally from a previous good evaluation (the "source") toward the failing one (the "target"). Further details about the algorithm employed by Dakota are supplied in the User's Manual [5].

7.5.5 deactivate

- [Keywords Area](#)
- [interface](#)
- [deactivate](#)

Deactivate Dakota interface features for simplicity or efficiency

Specification

Alias: none

Argument(s): none

Default: Active set vector control, function evaluation cache, and restart file features are active

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		active_set_vector	Deactivate the Active Set Vector
	Optional		evaluation_cache	Do not retain function evaluation history in memory
	Optional		strict_cache_equality	Do not require strict cache equality when finding duplicates
	Optional		restart_file	Deactivate writing to the restart file

Description

The optional `deactivate` specification block allows a user to deactivate interface features in order to simplify interface development, increase execution speed, and/or reduce memory and disk requirements. Any or all of these features may be specified concurrently.

- Active set vector (ASV) control: deactivate so that Dakota expects the same response data (all functions, gradients, Hessian) back from the simulation on every evaluation, instead of only those components required by the method for this particular function evaluation.
- Function evaluation cache: save memory by not caching the function evaluation history. May result in additional (duplicate) function evaluations.

- Strict cache equality: allow a relaxed tolerance when detecting duplicate function evaluations. Can be useful when importing data or restarting across machines.
- Restart file: improve efficiency and eliminate restart file storage at the risk of not being able to recover a failed or partial Dakota study.

7.5.5.1 active_set_vector

- [Keywords Area](#)
- [interface](#)
- [deactivate](#)
- [active_set_vector](#)

Deactivate the Active Set Vector

Specification

Alias: none

Argument(s): none

Description

Allows the user to turn off any variability in ASV values so that active set logic can be omitted in the user's simulation interface. This option trades some efficiency for simplicity in interface development.

The default behavior is to request the minimum amount of data required by an algorithm at any given time, which implies that the ASV values may vary from one function evaluation to the next. Since the user's interface must return the data set requested by the ASV values, this interface must contain additional logic to account for any variations in ASV content.

Deactivating this ASV control causes Dakota to always request a "full" data set (the full function, gradient, and Hessian data that is available from the interface as specified in the responses specification) on each function evaluation.

For example, if ASV control has been deactivated and the responses section specifies four response functions, analytic gradients, and no Hessians, then the ASV on every function evaluation will be { 3 3 3 3 }, regardless of what subset of this data is currently needed. While wasteful of computations in many instances, this simplifies the interface and allows the user to return the same data set on every evaluation. Conversely, if ASV control is active (the default behavior), then the ASV requests in this example might vary from { 1 1 1 1 } to { 2 0 0 2 }, etc., according to the specific data needed on a particular function evaluation. This will require the user's interface to read the ASV requests and perform the appropriate logic in conditionally returning only the data requested.

Usage Tips

- In general, the default ASV behavior is recommended for the sake of computational efficiency, unless interface development time is a critical concern.
- Whether active or inactive, the data returned to Dakota from the user's interface must match the ASV passed in, or else a response recovery error will result. However, when the ASV control is deactivated, the ASV values are invariant and need not be checked on every evaluation.
- Deactivating the ASV control can have a positive effect on load balancing for parallel Dakota executions. Thus, there is significant overlap in this ASV control option with speculative gradients. There is also overlap with the mode override approach used with certain optimizers to combine individual value, gradient, and Hessian requests.

7.5.5.2 `evaluation_cache`

- [Keywords Area](#)
- [interface](#)
- [deactivate](#)
- [evaluation_cache](#)

Do not retain function evaluation history in memory

Specification

Alias: none

Argument(s): none

Description

Do not retain the complete function evaluation history in memory.

This can be important for reducing memory requirements in large-scale applications (i.e., applications with a large number of variables or response functions) and for eliminating the overhead of searching for duplicates within the function evaluation cache prior to each new function evaluation (e.g., for improving speed in problems with 1000's of inexpensive function evaluations or for eliminating overhead when performing timing studies).

However, the downside is that unnecessary computations may be performed since duplication in function evaluation requests may not be detected. For this reason, this option is not recommended when function evaluations are costly.

Note: duplication detection within Dakota can be deactivated, but duplication detection features within specific optimizers may still be active.

7.5.5.3 `strict_cache_equality`

- [Keywords Area](#)
- [interface](#)
- [deactivate](#)
- [strict_cache_equality](#)

Do not require strict cache equality when finding duplicates

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		cache_tolerance	Specify tolerance when identifying duplicate function evaluations

Description

By default, Dakota's evaluation cache and restart capabilities are based on strict binary equality. This provides a performance advantage, as it permits a hash-based data structure to be used to search the evaluation cache. However, deactivating strict equality may prevent cache misses, which can occur when attempting to use a restart file on a machine different from the one on which it was generated.

7.5.5.3.1 cache_tolerance

- [Keywords Area](#)
- [interface](#)
- [deactivate](#)
- [strict_cache_equality](#)
- [cache_tolerance](#)

Specify tolerance when identifying duplicate function evaluations

Specification

Alias: none

Argument(s): REAL

Description

Described on parent page

7.5.5.4 restart_file

- [Keywords Area](#)
- [interface](#)
- [deactivate](#)
- [restart_file](#)

Deactivate writing to the restart file

Specification

Alias: none

Argument(s): none

Description

Eliminate the output of each new function evaluation to the binary restart file. This can increase speed and reduce disk storage requirements, but at the expense of a loss in the ability to recover and continue a run that terminates prematurely (e.g., due to a system crash or network problem).

Usage Tips

- This option is not recommended when function evaluations are costly or prone to failure.

- Using the `deactivate restart_file` specification will result in a zero length restart file with the default name `dakota.rst`.

7.5.6 batch

- [Keywords Area](#)
- [interface](#)
- [batch](#)

Perform evaluations in batches

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Default: sequential interface usage

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		size	Limit the number of evaluations in a batch

Description

When the optional `batch` keyword is used, Dakota performs evaluations in batches. In batch mode, Dakota writes the parameters for multiple (a batch of) evaluations to a single file, invokes the analysis driver once for the whole batch, and expects to find results for the entire batch in a single file after the analysis driver has exited. Batch mode may be useful when a user desires to take greater control over job management. For example, the analysis driver can perform the evaluations in the batch in any sequence or in concurrent sub-batches.

The names of the parameters file and results file are provided as command line arguments to the analysis driver, just as in a conventional, non-batch evaluation. By default, all currently available evaluations will be performed in a single batch, but the batch size can be limited using the [size](#) keyword.

Batch mode has a few important limitations.

- Only one batch at a time may be executed. Asynchronous execution of multiple concurrent batches is not supported.
- No [input_filter](#) or [output_filter](#) is permitted.
- Only one `analysis_driver` is allowed.
- `failure_capture` modes are restricted to [abort](#) and [recover](#).

Some of these restrictions may be lifted in future Dakota releases.

File Formats

A batch parameters file written by Dakota is simply a concatenated set of parameters files for the set of evaluations, either in [aprepro](#) or default Dakota format.

The batch results file is also a concatenated set of results files for the individual evaluations. However, because Dakota's results file format is not as rich as its parameters file format, evaluations in the batch results file must be separated by a line that begins with the '#' character.

The order of evaluations in the batch results file must match the order in the batch parameters file.

Tagging and Work Directories

When Dakota's [work_directory](#) feature is enabled, one directory is created per batch. If [file_tag](#) or [directory_tag](#) is used, parameters/results files and work directories are tagged with a batch Id, which is an integer that begins with 1 and increments for each new batch.

7.5.6.1 size

- [Keywords Area](#)
- [interface](#)
- [batch](#)
- [size](#)

Limit the number of evaluations in a batch

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Default: local: unlimited batch size, hybrid: zero batch size

Description

When `batch` execution is enabled, the default behavior is to add all available evaluations to the current batch. The `size` keyword limits the number of evaluations in a batch.

7.5.7 asynchronous

- [Keywords Area](#)
- [interface](#)
- [asynchronous](#)

Specify local evaluation or analysis concurrency

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Default: synchronous interface usage

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		evaluation_-concurrency	Determine how many concurrent evaluations Dakota will schedule
	Optional		local_evaluation_-scheduling	Control how local asynchronous jobs are scheduled
	Optional		analysis_-concurrency	Limit the number of analysis drivers within an evaluation that Dakota will schedule

Description

The optional `asynchronous` keyword specifies use of asynchronous protocols (i.e., background system calls, nonblocking forks, POSIX threads) when evaluations or analyses are invoked. Evaluation and analysis concurrency can be independently controlled, as can the scheduling mode (static vs. dynamic) of the local evaluations.

Default Behavior

- when running Dakota on a single processor in `asynchronous` mode, the default concurrency of evaluations and analyses is all concurrency that is available. The `evaluation_concurrency` and `analysis_concurrency` specifications can be used to limit this concurrency in order to avoid machine overload or usage policy violation.
- when running Dakota on multiple processors in message passing mode, the default concurrency of evaluations and analyses on each of the servers is one (i.e., the parallelism is exclusively that of the message passing). With the `evaluation_concurrency` and `analysis_concurrency` specifications, a hybrid parallelism can be selected through combination of message passing parallelism with asynchronous parallelism on each server.

7.5.7.1 `evaluation_concurrency`

- [Keywords Area](#)
- [interface](#)
- [asynchronous](#)

- [evaluation_concurrency](#)

Determine how many concurrent evaluations Dakota will schedule

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Default: local: unlimited concurrency, hybrid: no concurrency

Description

When `asynchronous` execution is enabled, the default behavior is to launch all available evaluations simultaneously. The `evaluation_concurrency` keyword can be used to limit the number of concurrent evaluations.

7.5.7.2 local_evaluation_scheduling

- [Keywords Area](#)
- [interface](#)
- [asynchronous](#)
- [local_evaluation_scheduling](#)

Control how local asynchronous jobs are scheduled

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Default: dynamic

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Required(<i>Choose One</i>)	Scheduling Mode (Group 1)	dynamic	Dynamic local scheduling (sequential)
			static	Static local scheduling (tiled)

Description

When performing asynchronous local evaluations, the `local_evaluation_scheduling` keyword controls how new evaluation jobs are dispatched when one completes.

The two options are:

- `dynamic`
- `static`

If the `local_evaluation_scheduling` is specified as `dynamic` (the default), each completed evaluation will be replaced by the next in the local evaluation queue.

If `local_evaluation_scheduling` is specified as `static`, each completed evaluation will be replaced by an evaluation number that is congruent modulo the `evaluation_concurrency`. This is helpful for relative node scheduling as described in `dakota/share/dakota/examples/parallelism`. For example, assuming only asynchronous local concurrency (no MPI), if the local concurrency is 6 and job 2 completes, it will be replaced with job 8.

For the case of hybrid parallelism, static local scheduling results in evaluation replacements that are modulo the total capacity, defined as the product of the evaluation concurrency and the number of evaluation servers. Both of these cases can result in idle processors if runtimes are non-uniform, so the default dynamic scheduling is preferred when relative node scheduling is not required.

7.5.7.2.1 `dynamic`

- [Keywords Area](#)
- [interface](#)
- [asynchronous](#)
- [local_evaluation_scheduling](#)
- [dynamic](#)

Dynamic local scheduling (sequential)

Specification

Alias: none

Argument(s): none

Description

If the `local_evaluation_scheduling` is specified as `dynamic` (the default), each completed evaluation will be replaced by the next in the local evaluation queue.

7.5.7.2.2 `static`

- [Keywords Area](#)
- [interface](#)

- [asynchronous](#)
- [local_evaluation_scheduling](#)
- [static](#)

Static local scheduling (tiled)

Specification

Alias: none

Argument(s): none

Description

If `local_evaluation_scheduling` is specified as `static`, each completed evaluation will be replaced by an evaluation number that is congruent modulo the `evaluation_concurrency`. This is helpful for relative node scheduling as described in `dakota/share/dakota/examples/parallelism`. For example, assuming only asynchronous local concurrency (no MPI), if the local concurrency is 6 and job 2 completes, it will be replaced with job 8.

For the case of hybrid parallelism, static local scheduling results in evaluation replacements that are modulo the total capacity, defined as the product of the evaluation concurrency and the number of evaluation servers. Both of these cases can result in idle processors if runtimes are non-uniform, so the default dynamic scheduling is preferred when relative node scheduling is not required.

7.5.7.3 `analysis_concurrency`

- [Keywords Area](#)
- [interface](#)
- [asynchronous](#)
- [analysis_concurrency](#)

Limit the number of analysis drivers within an evaluation that Dakota will schedule

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Default: local: unlimited concurrency, hybrid: no concurrency

Description

When `asynchronous` execution is enabled and each evaluation involves multiple analysis drivers, then the default behavior is to launch all drivers simultaneously. The `analysis_concurrency` keyword can be used to limit the number of concurrently run drivers.

7.5.8 `evaluation_servers`

- [Keywords Area](#)
- [interface](#)
- [evaluation_servers](#)

Specify the number of evaluation servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Default: automatic (see discussion)

Description

The optional `evaluation_servers` specification supports user override of the automatic parallel configuration for the number of evaluation servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the evaluation parallelism level. Refer to `ParallelLibrary` and the `Parallel Computing` chapter of the Users Manual [5] for additional information.

7.5.9 `evaluation_scheduling`

- [Keywords Area](#)
- [interface](#)
- [evaluation_scheduling](#)

Specify the scheduling of concurrent evaluations when Dakota is run in parallel

Specification

Alias: none

Argument(s): none

Default: automatic (see discussion)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Server Mode (Group 1)	master	Specify a dedicated master partition for parallel evaluation scheduling
			peer	Specify a peer partition for parallel evaluation scheduling

Description

When Dakota is run in parallel, the partition type and scheduling for the evaluation servers are determined automatically. If these settings are undesirable, they may be overridden by the user using the `evaluation_scheduling` keyword.

7.5.9.1 master

- [Keywords Area](#)
- [interface](#)
- [evaluation_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel evaluation scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the evaluation servers. This reduces the number of processors available to create servers by 1.

7.5.9.2 peer

- [Keywords Area](#)
- [interface](#)
- [evaluation_scheduling](#)
- [peer](#)

Specify a peer partition for parallel evaluation scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Scheduling Mode (Group 1)	Dakota Keyword	Dakota Keyword Description
			dynamic	Specify dynamic scheduling in a peer partition when Dakota is run in parallel.
			static	Specify static scheduling in a peer partition when Dakota is run in parallel.

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to evaluation servers. The scheduling, `static` or `dynamic`, must also be specified.

7.5.9.2.1 dynamic

- [Keywords Area](#)
- [interface](#)
- [evaluation_scheduling](#)
- [peer](#)
- [dynamic](#)

Specify dynamic scheduling in a peer partition when Dakota is run in parallel.

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Default: dynamic (see discussion)

Description

In `dynamic` scheduling, evaluations are assigned to servers as earlier evaluations complete. Dynamic scheduling is advantageous when evaluations are of uneven duration.

7.5.9.2.2 static

- [Keywords Area](#)
- [interface](#)
- [evaluation_scheduling](#)
- [peer](#)
- [static](#)

Specify static scheduling in a peer partition when Dakota is run in parallel.

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

In `static` scheduling, all available evaluations are assigned to servers in a predetermined fashion. Each completed evaluation is replaced with one congruent modulo the evaluation concurrency. For example, with 6 servers, eval number 2 will be replaced by eval number 8.

7.5.10 processors_per_evaluation

- [Keywords Area](#)
- [interface](#)
- [processors_per_evaluation](#)

Specify the number of processors per evaluation server when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Default: automatic (see discussion)

Description

The optional `processors_per_evaluation` specification supports user override of the automatic parallel configuration for the number of processors in each evaluation server. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired server size at the evaluation parallelism level. Refer to `ParallelLibrary` and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.5.11 `analysis_servers`

- [Keywords Area](#)
- [interface](#)
- [analysis_servers](#)

Specify the number of analysis servers when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): INTEGER

Default: automatic (see discussion)

Description

The optional `analysis_servers` specification supports user override of the automatic parallel configuration for the number of analysis servers. That is, if the automatic configuration is undesirable for some reason, the user can enforce a desired number of partitions at the analysis parallelism level. Refer to `ParallelLibrary` and the Parallel Computing chapter of the Users Manual [5] for additional information.

7.5.12 analysis_scheduling

- [Keywords Area](#)
- [interface](#)
- [analysis_scheduling](#)

Specify the scheduling of concurrent analyses when Dakota is run in parallel

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Default: automatic (see discussion)

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Scheduling Mode (Group 1)	master	Specify a dedicated master partition for parallel analysis scheduling
			peer	Specify a peer partition for parallel analysis scheduling

Description

When Dakota is run in parallel, the partition type for the analysis servers is determined automatically. If this setting is undesirable, it may be overridden by the user using the `analysis_scheduling` keyword.

7.5.12.1 master

- [Keywords Area](#)
- [interface](#)
- [analysis_scheduling](#)
- [master](#)

Specify a dedicated master partition for parallel analysis scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a dedicated master partition. In a dedicated master partition, one processor (the "master") dynamically schedules work on the analysis servers. This reduces the number of processors available to create servers by 1.

7.5.12.2 peer

- [Keywords Area](#)
- [interface](#)
- [analysis_scheduling](#)
- [peer](#)

Specify a peer partition for parallel analysis scheduling

Topics

This keyword is related to the topics:

- [concurrency_and_parallelism](#)

Specification

Alias: none

Argument(s): none

Description

This option overrides the Dakota parallel automatic configuration, forcing the use of a peer partition. In a peer partition, all processors are available to be assigned to analysis servers. Note that unlike the case of `evaluation_scheduling`, it is not possible to specify `static` or `dynamic`.

7.6 responses

- [Keywords Area](#)
- [responses](#)

Description of the model output data returned to Dakota upon evaluation of an interface.

Topics

This keyword is related to the topics:

- [block](#)

Specification

Alias: none

Argument(s): none

Child Keywords:

Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
Optional		id_responses	Name the responses block; helpful when there are multiple
Optional		descriptors	Labels for the responses
Required (Choose One)	Response Type (Group 1)	objective_functions	Response type suitable for optimization
		calibration_terms	Response type suitable for calibration or least squares
		response_functions	Generic response type
Required (Choose One)	Gradient Type (Group 2)	no_gradients	Gradients will not be used
		analytic_gradients	Analysis driver will return gradients
		mixed_gradients	Gradients are needed and will be obtained from a mix of numerical and analytic sources
		numerical_gradients	Gradients are needed and will be approximated by finite differences
Required (Choose One)	Hessian Type (Group 3)	no_hessians	Hessians will not be used
		numerical_hessians	Hessians are needed and will be approximated by finite differences

		quasi_hessians	Hessians are needed and will be approximated by secant updates (BFGS or SR1) from a series of gradient evaluations
		analytic_hessians	Hessians are needed and are available directly from the analysis driver
		mixed_hessians	Hessians are needed and will be obtained from a mix of numerical, analytic, and "quasi" sources
	Optional	metadata	(Experimental) Labels for floating point response metadata

Description

The `responses` specification in a Dakota input file indicates the types of data that can be returned by an interface when invoked during Dakota's execution. The specification includes three groups and two optional keywords.

The response type group indicates the type and number of responses expected by Dakota. It must be one of three types:

1. Optimization: objective and constraint functions
2. Calibration: calibration (least squares) terms and constraint functions
3. Uncertainty Quantification: generic response functions

The response type specified should be consistent with the iterative technique called for in the method specification. Certain general-purpose iterative techniques, such as parameter studies and design of experiments methods, can be used with any of these response types.

The gradient type group indicates the availability of first derivatives (gradient vectors) for the response functions.

The gradient specification also links back to the iterative method used. Gradients commonly are needed when the iterative study involves gradient-based optimization, local reliability analysis for uncertainty quantification, or local sensitivity analysis. They can optionally be used to build some types of surrogate models.

The Hessian type group specifies the availability of second derivatives (Hessian matrices) for the response functions.

Hessian availability for the response functions is similar to the gradient availability specifications, with the addition of support for "quasi-Hessians". The Hessian specification also links back to the iterative method in use; Hessians commonly would be used in gradient-based optimization by full Newton methods or in reliability analysis with second-order limit state approximations or second-order probability integrations.

Examples

Several examples follow. The first example shows an optimization data set containing an objective function and two nonlinear inequality constraints. These three functions have analytic gradient availability and no Hessian availability.

```
responses
  objective_functions = 1
    nonlinear_inequality_constraints = 2
  analytic_gradients
  no_hessians
```

The next example shows a typical specification for a calibration data set. The six residual functions will have numerical gradients computed using the dakota finite differencing routine with central differences of 0.1% (plus/minus delta relative to current variables value = .001*value).

```
responses
  calibration_terms = 6
  numerical_gradients
    method_source dakota
    interval_type central
    fd_gradient_step_size = .001
  no_hessians
```

The last example shows a generic specification that could be used with a nondeterministic sampling iterator. The three response functions have no gradient or Hessian availability; therefore, only function values will be used by the iterator.

```
responses
  response_functions = 3
  no_gradients
  no_hessians
```

Parameter study and design of experiments iterators are not restricted in terms of the response data sets which may be catalogued; they may be used with any of the function specification examples shown above.

Theory

Responses specify the total data set that is available for use by the method over the course of iteration. This is distinguished from the data subset described by an active set vector (see Dakota File Data Formats in the Users Manual [Adams et al., 2010]) indicating the particular subset of the response data needed for a particular function evaluation. Thus, the responses specification is a broad description of the data to be used during a study whereas the active set vector indicates the subset currently needed.

7.6.1 id_responses

- [Keywords Area](#)
- [responses](#)
- [id_responses](#)

Name the responses block; helpful when there are multiple

Topics

This keyword is related to the topics:

- [block_identifier](#)

Specification

Alias: none

Argument(s): STRING

Default: use of last responses parsed

Description

The optional `id_responses` keyword accepts a string that uniquely identifies this responses block. A model can then use these responses by specifying the same string in its `responses_pointer` specification.

Default Behavior

If the `id_responses` specification is omitted, a particular responses specification will be used by a model only if that model does not include an `responses_pointer` and the responses block was the last (or only) one parsed.

Usage Tips

- It is a best practice to always use explicit responses IDs and pointers to avoid confusion.
- If only one responses block exists, then `id_responses` can be safely omitted from the responses block (and `responses_pointer` omitted from the model specification(s)), since there is no ambiguity.

Examples

For example, a model specification including

```
model
  responses_pointer = 'R1'
```

will link to a response set with

```
id_responses = 'R1'
```

7.6.2 descriptors

- [Keywords Area](#)
- [responses](#)
- [descriptors](#)

Labels for the responses

Specification

Alias: response_descriptors

Argument(s): STRINGLIST

Default: root strings plus numeric identifiers

Description

A list of strings which identify the responses. These are used in console and tabular output. Response descriptors are ordered by primary response functions (objective, calibration, or response functions), followed by inequality, then equality constraints, if present.

Default Behavior

The default descriptor strings use a response type-dependent root string plus a one-based numeric identifier:

- Objective functions: `obj_fn_i`
- Calibration terms: `least_sq_term_i`
- Nonlinear inequality constraints: `nln_ineq_con_i`
- Nonlinear equality constraints: `nln_eq_con_i`
- Response functions: `response_fn_`

Expected Output Dakota will label the various response functions in console and tabular output.

Usage Tips When specifying descriptors for scalar and/or field responses, include as many descriptors as top-level scalar + field responses, e.g.,

```
responses
descriptors 'scalar1' 'scalar2' 'scalar3' 'field1' 'field2'
response_functions 5
  scalar_responses 3
  field_responses 2
  field_lengths 42 24
```

Dakota will append a numeric identifier for each field entry, for a total of 42 `field1_j` and 24 `field2_k` in this example.

Examples

```
responses
descriptors 'cost' 'impact' 'safety'
objective_functions 2
nonlinear_inequality_constraints 1
```

7.6.3 objective_functions

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)

Response type suitable for optimization

Specification

Alias: `num_objective_functions`

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		sense	Whether to minimize or maximize each objective function

	Optional	primary_scale_types	How to scale each objective function
	Optional	primary_scales	Characteristic values to scale each objective function
	Optional	weights	Specify weights for each objective function
	Optional	nonlinear_inequality_constraints	Group to specify nonlinear inequality constraints
	Optional	nonlinear_equality_constraints	Group to specify nonlinear equality constraints
	Optional	scalar_objectives	Number of scalar objective functions
	Optional	field_objectives	Number of field objective functions

Description

Specifies the number (1 or more) of objective functions f_j returned to Dakota for use in the general optimization problem formulation:

$$\begin{aligned}
 \text{minimize:} \quad & f(\mathbf{x}) = \sum_j w_j f_j \\
 & \mathbf{x} \in \mathfrak{X}^n \\
 \text{subject to:} \quad & \mathbf{g}_L \leq \mathbf{g}(\mathbf{x}) \leq \mathbf{g}_U \\
 & \mathbf{h}(\mathbf{x}) = \mathbf{h}_t \\
 & \mathbf{a}_L \leq \mathbf{A}_i \mathbf{x} \leq \mathbf{a}_U \\
 & \mathbf{A}_e \mathbf{x} = \mathbf{a}_t \\
 & \mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U
 \end{aligned}$$

Unless [sense](#) is specified, Dakota will minimize the objective functions.

The keywords [nonlinear_inequality_constraints](#) and [nonlinear_equality_constraints](#) specify the number of nonlinear inequality constraints g , and nonlinear equality constraints h , respectively. When interfacing to external applications, the responses must be returned to Dakota in this order in the [results_file](#) :

1. objective functions
2. [nonlinear_inequality_constraints](#)
3. [nonlinear_equality_constraints](#)

An optimization problem's linear constraints are provided to the solver at startup only and do not need to be included in the data returned on every function evaluation. Linear constraints are therefore specified in the [variables](#) block through the [linear_inequality_constraint_matrix](#) A_i and [linear_equality_constraint_matrix](#) A_e .

Lower and upper bounds on the design variables x are also specified in the [variables](#) block.

The optional keywords relate to scaling the objective functions (for better numerical results), formulating the problem as minimization or maximization, and dealing with multiple objective functions through [weights](#) w . If scaling is used, it is applied before multi-objective weighted sums are formed, so, e.g, when both weighting and characteristic value scaling are present the ultimate objective function would be:

$$f = \sum_{j=1}^n w_j \frac{f_j}{s_j}$$

See Also

These keywords may also be of interest:

- [calibration_terms](#)
- [response_functions](#)
- [method](#)
- [variables](#)

7.6.3.1 sense

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [sense](#)

Whether to minimize or maximize each objective function

Specification

Alias: none

Argument(s): STRINGLIST

Default: vector values = 'minimize'

Description

The `sense` keyword is used to declare whether each objective function should be minimized or maximized. The argument options are:

- "minimization" (can be abbreviated to "min")
- "maximization" (can be abbreviated to "max")

If a single string is specified it will apply to each objective function. Otherwise, the number of strings should either be equal to the number of objective functions ([objective_functions](#)). Thus, when scalar and/or field responses are specified, the number of `sense` strings must equal the number of scalars plus the number of fields, *not* the total elements in the fields.

7.6.3.2 `primary_scale_types`

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [primary_scale_types](#)

How to scale each objective function

Specification

Alias: `objective_function_scale_types`

Argument(s): STRINGLIST

Default: no scaling

Description

Each string in `primary_scale_types` indicates the scaling type for each objective function. They only have effect when the associated method specifies `scaling`.

The options are:

- `'value'` - characteristic value by which response functions will be divided. If this is chosen, then [primary_scales](#) must also be specified; `'value'` is assumed if scales are given without `scale_types`
- `'log'` - logarithmic scaling (can be used together with [primary_scales](#), which can be helpfully used to negate values prior to log transformation)

(Automatic scaling is disallowed due to lack of bounds or targets for primary responses.)

If a single string is specified it will apply to all `objective_functions`. Otherwise the length of `primary_scale_types` must have length equal to [objective_functions](#). Thus, when scalar and/or field responses are specified, the number of `primary_scale_types` must equal the number of scalars plus the number of fields, *not* the total elements in the fields.

Usage Tips:

When scaling is active, it is applied to objective functions prior to any [weights](#) and multi-objective sum formation.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.3 `primary_scales`

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [primary_scales](#)

Characteristic values to scale each objective function

Specification

Alias: `objective_function_scales`

Argument(s): REALLIST

Default: 1.0 (no scaling)

Description

Each real value in `primary_scales` is a nonzero characteristic value to be used in scaling each objective function. They only have effect when the associated method specifies `scaling`.

This keyword is required for [primary_scale_types](#) of 'value' and optional for 'log'. When specified in conjunction with log, scale values are applied prior to the logarithm, to permit log scaling of strictly negative response functions.

Length: When specified, `primary_scales` must have length one of:

- One (the single value will be applied to each objective function); or
- Number of response groups ([objective_functions](#)), so the number of scalars plus the number of fields when fields are present; or
- Total number of response elements, so the number of scalar responses plus the sum of the lengths of the fields.

Usage Tips:

When `primary_scales` are specified, but not [primary_scale_types](#), the scaling type is assumed to be 'value' for all primary response functions.

Use scale values of 1.0 to selectively avoid scaling a subset of response functions.

When scaling is active, it is applied to objective functions prior to any [weights](#) and multi-objective sum formation. See the equations in [objective_functions](#).

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.4 weights

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [weights](#)

Specify weights for each objective function

Specification

Alias: `multi_objective_weights`

Argument(s): REALLIST

Default: equal weights

Description

For multi-objective optimization problems (where the number of objective functions is greater than 1), then a `weights` specification provides a simple weighted-sum approach to combining multiple objectives into a single objective:

$$f = \sum_{i=1}^n w_i f_i$$

Length: The `weights` must have length equal to [objective_functions](#). Thus, when scalar and/or field responses are specified, the number of `weights` must equal the number of scalars plus the number of fields, *not* the total elements in the fields.

Default Behavior If weights are not specified, then each response is given equal weighting:

$$f = \sum_{i=1}^n \frac{f_i}{n}$$

where, in both of these cases, a "minimization" [sense](#) will retain a positive weighting for a minimizer and a "maximization" sense will apply a negative weighting.

Usage Tips:

Weights are applied as multipliers, scales as characteristic values / divisors.

When scaling is active, it is applied to objective functions prior to any `weights` and multi-objective sum formation. See the equations in [objective_functions](#).

7.6.3.5 nonlinear_inequality_constraints

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_inequality_constraints](#)

Group to specify nonlinear inequality constraints

Topics

This keyword is related to the topics:

- [nonlinear_constraints](#)

Specification

Alias: num_nonlinear_inequality_constraints

Argument(s): INTEGER

Default: 0

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	lower_bounds	Specify minimum values
	Optional	upper_bounds	Specify maximum values
	Optional	scale_types	How to scale each nonlinear constraint
	Optional	scales	Characteristic values to scale each nonlinear constraint

Description

Specifies the number of nonlinear inequality constraint functions returned by the interface.

The `lower_bounds` and `upper_bounds` specifications provide the lower and upper bounds for 2-sided nonlinear inequalities of the form

$$g_l \leq g(x) \leq g_u$$

When constraint bounds are not specified, the problem is assumed to have one-sided inequalities bounded above by zero:

$$g(x) \leq 0.0.$$

This provides backwards compatibility with previous Dakota versions.

In a user bounds specification, any upper bound values greater than `+bigRealBoundSize` (1.e+30, as defined in `Minimizer`) are treated as +infinity and any lower bound values less than `-bigRealBoundSize` are treated as -infinity. This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since `-DBL_MAX < -bigRealBoundSize`). The same approach is used for nonexistent linear inequality bounds and for nonexistent design variable bounds.

The `scale_types` and `scales` keywords are related to scaling of $g(x)$. See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.5.1 lower_bounds

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_inequality_constraints](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: `nonlinear_inequality_lower_bounds`

Argument(s): REALLIST

Default: vector values = -infinity

Description

Specify minimum values

7.6.3.5.2 upper_bounds

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_inequality_constraints](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: nonlinear_inequality_upper_bounds

Argument(s): REALLIST

Default: vector values = 0 .

Description

Specify maximum values

7.6.3.5.3 scale_types

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_inequality_constraints](#)
- [scale_types](#)

How to scale each nonlinear constraint

Specification

Alias: nonlinear_inequality_scale_types

Argument(s): STRINGLIST

Default: no scaling

Description

Each string in `scale_types` indicates the scaling type for each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

The options are:

- `'value'` - characteristic value by which nonlinear constraint values will be divided. If this is chosen, then `scales` must also be specified; `'value'` is assumed if `scales` are given without `scale_types`
- `'auto'` - automatic scaling based on bounds (inequalities) or targets (equalities)

- 'log' - logarithmic scaling (can be used together with `scales`, which can be helpfully used to negate values prior to log transformation)

If a single string is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a string must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.5.4 scales

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_inequality_constraints](#)
- [scales](#)

Characteristic values to scale each nonlinear constraint

Specification

Alias: `nonlinear_inequality_scales`

Argument(s): REALLIST

Default: 1.0 (no scaling)

Description

Each real value in `scales` is a nonzero characteristic value to be used in scaling each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

This keyword is required for `scale_types` of 'value' and optional for 'log'. When specified in conjunction with log, scale values are applied prior to the logarithm, to permit log scaling of strictly negative response functions.

If a single scale value is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a value must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

When `scales` are specified, but not `scale_types`, the scaling type is assumed to be 'value' for this set of nonlinear inequality (equality) constraints.

Use scale values of 1.0 to selectively avoid scaling a subset of response functions.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.6 nonlinear_equality_constraints

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_equality_constraints](#)

Group to specify nonlinear equality constraints

Topics

This keyword is related to the topics:

- [nonlinear_constraints](#)

Specification

Alias: num_nonlinear_equality_constraints

Argument(s): INTEGER

Default: 0

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		targets	Target values for the nonlinear equality constraint
	Optional		scale_types	How to scale each nonlinear constraint
	Optional		scales	Characteristic values to scale each nonlinear constraint

Description

Specifies the number of nonlinear equality constraint functions returned by the interface.

The `targets` specification provides the targets for nonlinear equalities of the form

$$h(x) = h_i$$

and the defaults for the equality targets enforce a value of 0. for each constraint

$$h(x) = 0.0$$

The `scale_types` and `scales` keywords are related to scaling of $h(x)$. See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.6.1 targets

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_equality_constraints](#)
- [targets](#)

Target values for the nonlinear equality constraint

Specification

Alias: `nonlinear_equality_targets`

Argument(s): REALLIST

Default: vector values = 0 .

Description

The `targets` specification provides the targets for nonlinear equalities of the form

$$g(x) = g_t$$

and the defaults for the equality targets enforce a value of 0 . 0 for each constraint:

$$g(x) = 0.0$$

7.6.3.6.2 `scale_types`

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_equality_constraints](#)
- [scale_types](#)

How to scale each nonlinear constraint

Specification

Alias: `nonlinear_equality_scale_types`

Argument(s): STRINGLIST

Default: no scaling

Description

Each string in `scale_types` indicates the scaling type for each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

The options are:

- `'value'` - characteristic value by which nonlinear constraint values will be divided. If this is chosen, then `scales` must also be specified; `'value'` is assumed if `scales` are given without `scale_types`
- `'auto'` - automatic scaling based on bounds (inequalities) or targets (equalities)
- `'log'` - logarithmic scaling (can be used together with `scales`, which can be helpfully used to negate values prior to log transformation)

If a single string is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a string must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.6.3 scales

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [nonlinear_equality_constraints](#)
- [scales](#)

Characteristic values to scale each nonlinear constraint

Specification

Alias: nonlinear_equality_scales

Argument(s): REALLIST

Default: 1.0 (no scaling)

Description

Each real value in `scales` is a nonzero characteristic value to be used in scaling each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

This keyword is required for `scale_types` of 'value' and optional for 'log'. When specified in conjunction with log, scale values are applied prior to the logarithm, to permit log scaling of strictly negative response functions.

If a single scale value is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a value must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

When `scales` are specified, but not `scale_types`, the scaling type is assumed to be 'value' for this set of nonlinear inequality (equality) constraints.

Use scale values of 1.0 to selectively avoid scaling a subset of response functions.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.3.7 scalar_objectives

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [scalar_objectives](#)

Number of scalar objective functions

Specification

Alias: num_scalar_objectives

Argument(s): INTEGER

Description

This keyword describes the number of scalar objective functions. It is meant to be used in conjunction with `field_objectives`, which describes the number of field objectives functions. The total number of objective functions, both scalar and field, is given by `objective_functions`. If only scalar objective functions are specified, it is not necessary to specify the number of scalar terms explicitly: one can simply say `objective_functions = 5` and get 5 scalar objectives. However, if there are three scalar objectives and 2 field objectives, then `objective_functions = 5` but `scalar_objectives = 3` and `field_objectives = 2`.

Objective functions are responses that are used with optimization methods in Dakota. Currently, each term in a field objective is added to the total objective function presented to the optimizer. For example, if you have one field objective with 100 terms (e.g. a time-temperature trace with 100 time points and 100 corresponding temperature points), the 100 temperature values will be added to create the overall objective.

See Also

These keywords may also be of interest:

- [field_objectives](#)

7.6.3.8 field_objectives

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [field_objectives](#)

Number of field objective functions

Specification

Alias: `num_field_objectives`

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		lengths	Lengths of field responses
	Optional		num_coordinates_per_field	Number of independent coordinates for field responses
	Optional		read_field_coordinates	Add context to data: flag to indicate that field coordinates should be read

Description

This keyword describes the number of field objective functions. A field function is a set of related response values collected over a range of independent coordinate values which may or may not be specified by the user. For example, voltage over time would be a field function, where voltage is the `field_objective` and time is the independent coordinate. Similarly, temperature over time and space would be a field response, where the independent coordinates would be both time and spatial coordinates such as (x,y) or (x,y,z), depending on the application. The main difference between scalar objectives and field objectives is that for field data, we plan to implement methods that take advantage of the correlation or relationship between the field values.

Note that if there is one `field_objective`, and it has length 100 (meaning 100 values), then the user's simulation code must return 100 values. Also, if there are both scalar and field objectives, the user should specify the number of scalar objectives as `scalar_objectives`. If there are only field objectives, it still is necessary to specify both `objective_functions = NN` and `field_objectives = NN`, where NN is the number of field objectives.

Objective functions are responses that are used with optimization methods in Dakota. Currently, each term in a field objective is added to the total objective function presented to the optimizer. For example, if you have one field objective with 100 terms (e.g. a time-temperature trace with 100 time points and 100 corresponding temperature points), the 100 temperature values will be added to create the overall objective.

See Also

These keywords may also be of interest:

- [scalar_objectives](#)

7.6.3.8.1 lengths

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [field_objectives](#)
- [lengths](#)

Lengths of field responses

Specification

Alias: none

Argument(s): INTEGERLIST

Description

This keyword describes the lengths of each field response. It is an integer vector of length `field_responses`. For example, if the `field_responses = 2`, an example would be `lengths = 50 200`, indicating that the first field response has 50 field elements but the second one has 200. The coordinate values (e.g. the independent variables) corresponding to these field responses are read in files labeled `response_descriptor.coords`.

See Also

These keywords may also be of interest:

- [field_responses](#)

7.6.3.8.2 num_coordinates_per_field

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [field_objectives](#)
- [num_coordinates_per_field](#)

Number of independent coordinates for field responses

Specification

Alias: none

Argument(s): INTEGERLIST

Description

This keyword describes the number of independent coordinates for each field response. It is an integer vector of length `field_responses`. For example, if the `field_responses = 2`, an example would be `num_coordinates_per_field = 2 1` means that the first field response has two sets of independent coordinates (perhaps x, y locations), but the second response only has one (for example, time where the field response is only dependent upon time). The actual coordinate values (e.g. the independent variables) corresponding to these field responses are defined in a file call `response_descriptor.coords`, where `response_descriptor` is the name of the individual field.

See Also

These keywords may also be of interest:

- [field_responses](#)

7.6.3.8.3 read_field_coordinates

- [Keywords Area](#)
- [responses](#)
- [objective_functions](#)
- [field_objectives](#)
- [read_field_coordinates](#)

Add context to data: flag to indicate that field coordinates should be read

Specification

Alias: none

Argument(s): none

Description

Field coordinates specify independent variables (e.g. spatial or temporal coordinates) upon which the field depends. For example, the voltage level above might be a function of time, so time is the field coordinate. If the user has field coordinates to read, they need to specify `read_field_coordinates`. The field coordinates will then be read from a file named `response_descriptor.coords`, where `response_descriptor` is the user-provided descriptor for the field response. The number of columns in the coords file should be equal to the number of field coordinates.

7.6.4 calibration_terms

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)

Response type suitable for calibration or least squares

Specification

Alias: `least_squares_terms` `num_least_squares_terms`

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		scalar_calibration_terms	Number of scalar calibration terms
	Optional Optional		field_calibration_terms primary_scales	Number of field calibration terms Characteristic values to scale each calibration term
	Optional		weights	Specify weights for each objective function
	Optional (<i>Choose One</i>)	Calibration Data (Group 1)	calibration_data	Supply field or mixed field/scalar calibration data

		calibration_data_file	Supply scalar calibration data only
	Optional	simulation_variance	Variance applied to simulation responses
	Optional	nonlinear_inequality_constraints	Group to specify nonlinear inequality constraints
	Optional	nonlinear_equality_constraints	Group to specify nonlinear equality constraints

Description

Responses for a calibration study are specified using `calibration_terms` and optional keywords for weighting/scaling, data, and constraints. In general when calibrating, Dakota automatically tunes parameters θ to minimize discrepancies or residuals between the model and the data:

$$R_i = y_i^{Model}(\theta) - y_i^{Data}.$$

Note that the problem specification affects what must be returned to Dakota in the `results_file` :

- If calibration data *is not specified*, then each of the calibration terms returned to Dakota through the `interface` is a residual R_i to be driven toward zero.
- If calibration data *is specified*, then each of the calibration terms returned to Dakota must be a response $y_i^{Model}(\theta)$, which Dakota will difference with the data in the specified data file.

Constraints

(See general problem formulation at [objective_functions](#).) The keywords `nonlinear_inequality_constraints` and `nonlinear_equality_constraints` specify the number of nonlinear inequality constraints g , and nonlinear equality constraints h , respectively. When interfacing to external applications, the responses must be returned to Dakota in this order in the `results_file` :

1. calibration terms
2. nonlinear inequality constraints
3. nonlinear equality constraints

An optimization problem's linear constraints are provided to the solver at startup only and do not need to be included in the data returned on every function evaluation. Linear constraints are therefore specified in the `variables` block through the `linear_inequality_constraint_matrix` A_i and `linear_equality_constraint_matrix` A_e .

Lower and upper bounds on the design variables x are also specified in the `variables` block.

Problem Transformations

Weighting or scaling calibration terms is often appropriate to account for measurement error or to condition the problem for easier solution. Weighting or scaling transformations are applied in the following order:

1. When present, observation error variance σ_i or full covariance Σ , optionally specified through `experiment_variance_type`, is applied to residuals first:

$$R_i^{(1)} = \frac{R_i}{\sigma_i} = \frac{y_i^{Model}(\theta) - y_i^{Data}}{\sigma_i}, \text{ or}$$

$$R^{(1)} = \Sigma^{-1/2} R = \Sigma^{-1/2} (y^{Model}(\theta) - y^{Data}),$$

resulting in the typical variance-weighted least squares formulation

$$\min_{\theta} R(\theta)^T \Sigma^{-1} R(\theta)$$

2. Any active scaling transformations are applied next, e.g., for characteristic value scaling:

$$R_i^{(2)} = \frac{R_i^{(1)}}{s_i}$$

3. Finally the optional weights are applied in a way that preserves backward compatibility:

$$R_i^{(3)} = \sqrt{w_i} R_i^{(2)}$$

so the ultimate least squares formulation, e.g., in a scaled and weighted case would be

$$f = \sum_{i=1}^n w_i \left(\frac{y_i^{Model} - y_i^{Data}}{s_i} \right)^2$$

Note that specifying observation error variance and weights are mutually exclusive in a calibration problem.

Theory

Dakota calibration terms are typically used to solve problems of parameter estimation, system identification, and model calibration/inversion. Local least squares calibration problems are most efficiently solved using special-purpose least squares solvers such as Gauss-Newton or Levenberg-Marquardt; however, they may also be solved using any general-purpose optimization algorithm in Dakota. While Dakota can solve these problems with either least squares or optimization algorithms, the response data sets to be returned from the simulator are different when using [objective_functions](#) versus [calibration_terms](#).

Least squares calibration involves a set of residual functions, whereas optimization involves a single objective function (sum of the squares of the residuals), i.e.,

$$f = \sum_{i=1}^n R_i^2 = \sum_{i=1}^n (y_i^{Model}(\theta) - y_i^{Data})^2$$

where f is the objective function and the set of R_i are the residual functions, most commonly defined as the difference between a model response and data. Therefore, function values and derivative data in the least squares case involve the values and derivatives of the residual functions, whereas the optimization case involves values and derivatives of the sum of squares objective function. This means that in the least squares calibration case, the user must return each of n residuals separately as a separate calibration term. Switching between the two approaches sometimes requires different simulation interfaces capable of returning the different granularity of response data required, although Dakota supports automatic recasting of residuals into a sum of squares for presentation to an optimization method. Typically, the user must compute the difference between the model results and the observations when computing the residuals. However, the user has the option of specifying the observational data (e.g. from physical experiments or other sources) in a file.

See Also

These keywords may also be of interest:

- [objective_functions](#)
- [response_functions](#)

7.6.4.1 scalar_calibration_terms

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [scalar_calibration_terms](#)

Number of scalar calibration terms

Specification

Alias: none

Argument(s): INTEGER

Description

This keyword describes the number of scalar calibration terms. It is meant to be used in conjunction with `field_calibration_terms`, which describes the number of field calibration terms. The total number of calibration terms, both scalar and field, is given by `calibration_terms`. If only scalar calibration terms are specified, it is not necessary to specify the number of scalar terms explicitly: one can simply say `calibration_terms = 5` and get 5 scalar terms. However, if there are three scalar terms and 2 field terms, then `calibration_terms = 5` but `scalar_calibration_terms = 3` and `field_calibration_terms = 2`.

Calibration terms are responses that are used with calibration methods in Dakota, such as least squares optimizers. Currently, each scalar term is added to the total sum-of-squares error function presented to the optimizer. However, each individual field value is added as well. For example, if you have one field calibration term with length 100 (e.g. a time - temperature trace with 100 time points and 100 temperature points), the 100 temperature values will be added to create the overall sum-of-squares error function used in calibration.

See Also

These keywords may also be of interest:

- [field_calibration_terms](#)

7.6.4.2 field_calibration_terms

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [field_calibration_terms](#)

Number of field calibration terms

Specification

Alias: none

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		lengths	Lengths of field responses
	Optional		num_coordinates_per_field	Number of independent coordinates for field responses
	Optional		read_field_coordinates	Add context to data: flag to indicate that field coordinates should be read

Description

This keyword describes the number of field calibration terms. A set of field calibration terms is a set of related response values collected over a range of independent coordinate values which may or may not be specified by the user. For example, voltage over time would be a field function, where voltage is the `field_objective` and time is the independent coordinate. Similarly, temperature over time and space would be a field response, where the independent coordinates would be both time and spatial coordinates such as (x,y) or (x,y,z), depending on the application. The main difference between scalar calibration terms and field calibration terms is that for field data, we plan to implement methods that take advantage of the correlation or relationship between the field values. For example, with calibration, if we want to calibrate parameters that result in a good model fit to a time-temperature curve, we may have to do some interpolation between the experimental data and the simulation data. That capability requires knowledge of the independent coordinates.

Note that if there is one `field_calibration_terms`, and it has length 100 (meaning 100 values), then the user's simulation code must return 100 values. Also, if there are both scalar and field calibration, the user should specify the number of scalar terms as `scalar_calibration_terms`. If there are only field calibration terms, it still is necessary to specify both `field_calibration_terms = NN` and `calibration_terms = NN`, where NN is the number of field calibration terms.

Calibration terms are responses that are used with calibration methods in Dakota, such as least squares optimizers. Currently, each scalar term is added to the total sum-of-squares error function presented to the optimizer. However, each individual field value is added as well. For example, if you have one field calibration term with length 100 (e.g. a time-temperature trace with 100 time points and 100 temperature points), the 100 temperature values will be added to create the overall sum-of-squares error function used in calibration. We have an initial capability to interpolate the field data from the user's simulation to the experimental data. For example, if the user has thermocouple readings at 20 time points, it will be an experimental field response with 20 time points and 20 temperature values. Dakota takes the 100 simulation time-temperature values (from the example above) and interpolates those to the 20 experimental points, to create 20 residual terms (simulation minus experimental data points) that will be used in calibration.

See Also

These keywords may also be of interest:

- [scalar_calibration_terms](#)

7.6.4.2.1 lengths

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [field_calibration_terms](#)

- [lengths](#)

Lengths of field responses

Specification

Alias: none

Argument(s): INTEGERLIST

Description

This keyword describes the lengths of each field response. It is an integer vector of length `field_responses`. For example, if the `field_responses = 2`, an example would be `lengths = 50 200`, indicating that the first field response has 50 field elements but the second one has 200. The coordinate values (e.g. the independent variables) corresponding to these field responses are read in files labeled `response_descriptor.coords`.

See Also

These keywords may also be of interest:

- [field_responses](#)

7.6.4.2.2 num_coordinates_per_field

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [field_calibration_terms](#)
- [num_coordinates_per_field](#)

Number of independent coordinates for field responses

Specification

Alias: none

Argument(s): INTEGERLIST

Description

This keyword describes the number of independent coordinates for each field response. It is an integer vector of length `field_responses`. For example, if the `field_responses = 2`, an example would be `num_coordinates_per_field = 2 1` means that the first field response has two sets of independent coordinates (perhaps x, y locations), but the second response only has one (for example, time where the field response is only dependent upon time). The actual coordinate values (e.g. the independent variables) corresponding to these field responses are defined in a file call `response_descriptor.coords`, where `response_descriptor` is the name of the individual field.

See Also

These keywords may also be of interest:

- [field_responses](#)

7.6.4.2.3 read_field_coordinates

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [field_calibration_terms](#)
- [read_field_coordinates](#)

Add context to data: flag to indicate that field coordinates should be read

Specification

Alias: none

Argument(s): none

Description

Field coordinates specify independent variables (e.g. spatial or temporal coordinates) upon which the field depends. For example, the voltage level above might be a function of time, so time is the field coordinate. If the user has field coordinates to read, they need to specify `read_field_coordinates`. The field coordinates will then be read from a file named `response_descriptor.coords`, where `response_descriptor` is the user-provided descriptor for the field response. The number of columns in the coords file should be equal to the number of field coordinates.

7.6.4.3 primary_scales

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [primary_scales](#)

Characteristic values to scale each calibration term

Specification

Alias: `calibration_term_scales` `least_squares_term_scales`

Argument(s): REALLIST

Default: 1.0 (no scaling)

Description

Each real value in `primary_scales` is a nonzero characteristic value to be used in scaling each calibration term. They only have effect when the associated method specifies `scaling`.

Length: When specified, `primary_scales` must have length one of:

- One (the single value will be applied to each primary response); or
- Number of response groups ([calibration_terms](#)), so the number of scalars plus the number of fields when fields are present; or
- Total number of response elements, so the number of scalar responses plus the sum of the lengths of the fields.

Usage Tips:

Calibration only supports value-based scaling. Use scale values of 1.0 to selectively avoid scaling a subset of response functions.

Weights are applied as multipliers, scales as characteristic values / divisors.

When scaling is active, it is applied to calibration terms after any residual formation (accounting for experimental data and optionally measurement error covariance), and before any weights. See the equations in [calibration_terms](#).

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.4.4 weights

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [weights](#)

Specify weights for each objective function

Specification

Alias: `calibration_weights` `least_squares_weights`

Argument(s): REALLIST

Default: equal weights

Description

Specifies relative emphasis through weights (multipliers) w on residual elements:

$$f = \sum_{i=1}^n w_i R_i^2 = \sum_{i=1}^n w_i (y_i^{Model} - y_i^{Data})^2$$

Length: The `weights` must have length equal to [calibration_terms](#). Thus, when scalar and/or field responses are specified, the number of `weights` must equal the number of scalars plus the number of fields, *not* the total elements in the fields.

Default Behavior If weights are not specified, then each residual is given equal weighting.

Usage Tips:

Weights are applied as multipliers, scales as characteristic values / divisors.

When scaling is active, it is applied to calibration terms after any residual formation (accounting for experimental data and optionally measurement error covariance), and before any weights. See the equations in [calibration_terms](#).

7.6.4.5 `calibration_data`

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)

Supply field or mixed field/scalar calibration data

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		data_directory	Specify a directory containing the calibration field data files used with the mixed case specification
	Optional		num_experiments	Add context to data: number of different experiments
	Optional		num_config_variables	Add context to data: number of configuration variables.
	Optional		experiment_variance_type	Add context to data: specify the type of experimental error
	Optional		scalar_data_file	Specify a scalar data file to complement field data files (mixed case)

	Optional	interpolate	Flag to indicate interpolation of simulation values.
--	-----------------	-----------------------------	--

Description

`calibration_data` specifies a keyword block that indicates that Dakota should read in experimental data for calibration. This block is primarily to support the reading of field calibration data. For simpler, scalar-only response cases, see [calibration_data_file](#). The user will typically specify the number of experiments, `num_experiments`. If this is not specified, it is assumed there is only one experiment.

Up to four types of data may be read. They are read from a collection of files, one per response descriptor, per experiment. In this discussion, DESC refers to the response descriptor and NUM to the experiment number.

1. **Values:** The scalar or field-valued response function values, e.g., temperature values, voltage levels. These are read from files named `DESC.NUM.dat` (one per response descriptor, per experiment), e.g, `volts.1.dat`, `volts.2.dat`. Scalar files will contain a single value, while field files will each contain a column of field response values. Without [interpolate](#) enabled, the lengths of these files must match those specified using the `responses-calibration_terms-field_calibration_terms-lengths` keyword.
2. **Coordinates:** Field coordinates specify independent variables (e.g., spatial or temporal coordinates) upon which the field depends. For example, the voltage might be a function of time, so time is the field coordinate. These are read from files named `DESC.NUM.coords`, each containing `num_coordinates_per_field` columns. The number of rows must be the same as in the values files described in the previous bullet.
3. **Variations:** If [experiment_variance_type](#) is specified, variance values are read from files `DESC.NUM.sigma`. Note that a single `experiment_variance_type` may be specified, or a unique `experiment_variance_type` per response descriptor (per scalar or field). If the `experiment_variance_type` is:
 - 'scalar': a single variance value will be read from the file.
 - 'diagonal' (field responses only): a column vector of variance values (length equal to the number of experimental data points) will be read from the file. The values are the variances of each field value for this descriptor.
 - 'matrix' (field responses only): a matrix of covariance values (square with size the number of experimental values) will be read from the file. The matrix is a full covariance matrix for the components of this field response. While covariance among entries in a field response may be specified, covariance among experiments is not permitted.
4. **Configuration variables:** specify the conditions corresponding to different experiments. When `responses-calibration_terms-calibration_data-num_config_variables` is specified, the configuration variable values for each experiment should be placed in a file named `experiment.NUM.config`, where the number of items in that config file are the `num_config_variables`. These variables are used as auxiliary state variables for the simulation (for example) and are not calibrated.

Attention: In versions of Dakota prior to 6.14, string-valued configuration variables were specified in data files with 0-based indices into the admissible values. As of Dakota 6.14, strings must be specified by value. For example a string-valued configuration variable for an experimental condition might appear in the file as `low_pressure` vs. `high_pressure`.

Aggregating scalar data: The above description is primarily relevant for field data (with files for field values, field coordinates, field variances). If the user also has scalar experimental data, it may be entered as described above, i.e., one file named `DESC.NUM.dat` per scalar response. However, an alternative is to provide the data for all scalar responses in aggregate in the simpler [scalar_data_file](#) format, with the number of rows of that file equal to the number of experiments. The scalar data file may be used in combination with the the separate field files described above.

Interpolation: One important feature of field data is the capability to interpolate between points in the field. For example, we may have simulation data at a set of responses y at time points t : (t_{s1}, y_{s1}) , (t_{s2}, y_{s2}) , etc. In this

example, t is the independent coordinate for the simulation, and the simulation time and response points are denoted with subscripts s_1, s_2, s_3, \dots . If the user has experimental data that is taken at different time points: $(t_{e1}, y_{e1}), (t_{e2}, y_{e2}), \dots$, it is necessary to interpolate the simulation data to provide estimates of the simulation response at the experimental time points to construct the residual terms (model - experiment) at the experimental time points. Dakota can perform 1-D interpolation. The user must specify the keyword `interpolate`, and also provide the field coordinates as well as field values for the experiment data.

If the `interpolate` keyword is not specified, Dakota will assume that the simulation field data and the experiment field data is taken at the same set of independent coordinate values and simply construct the difference between these field terms to create the set of residuals for the sum-of-squares calculation. When `interpolate` is specified, the simulation coordinates are assumed fixed and the same for each simulation. These simulation coordinates are provided in `DESC.coords`. However, the experiment coordinates for each experiment can be different, and are provided in the files numbered by experiment with the file names given by `DESC.NUM.coords`, as indicated above.

Examples

Consider this field example with three field-valued responses (volts as a function of 7 time points, amps as a function of 3 time points, and power as a function of 5 time points). The experiments each have one coordinate (time) per field. The experiments were conducted given values of two configuration variables. There are four experiments (note that there often is only one experiment).

```
responses
  descriptors = 'volts' 'amps' 'power'
  calibration_terms = 3

  #define field terms
  field_calibration_terms = 3
  lengths = 7 3 5
  num_coordinates_per_field = 1 1 1
  read_field_coordinates

  # specify the data
  calibration_data
  num_experiments = 4
  num_config_variables = 2
  experiment_variance_type 'diagonal' 'scalar' 'matrix'
```

The field data will be read from separate files:

1. **Values:** 7 values for voltage as a function of time will be read from each of `volts.1.dat ... volts.4.dat`, while the 3 values of amperage come from `amps.1.dat, ..., amps.4.dat` and the 5 values of power will come from `power.1.dat, ..., power.4.dat`.
2. **Coordinates:** The corresponding time values will be read from a single column vector in each of the files `volts.*.coords`, `amps.*.coords`, and `power.*.coords`.
3. **Variances:** Will be read per-experiment (no covariance among experiments is permitted). The variance information is depicted graphically in [experiment_variance_type](#)
 - Volts specifies 'diagonal' covariance, so a column vector of length 7 with the diagonal variances will be read from each of `volts.1.sigma, ..., volts.4.sigma`.
 - Amps specifies 'scalar' covariance, so a single variance value will read from each of `amps.1.sigma, ..., amps.4.sigma`.
 - Power specifies 'matrix' covariance, so a square 5 x 5 covariance matrix will be read from each of `power.1.sigma, ..., power.4.sigma`.
4. **Configuration variables:** Will be read from a set of files named `experiment.1.config, ..., experiment.4.config`.

7.6.4.5.1 data_directory

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [data_directory](#)

Specify a directory containing the calibration field data files used with the mixed case specification

Specification

Alias: none

Argument(s): STRING

Description**Default Behavior**

The default directory is the path from which the Dakota study is launched, i.e. the directory containing the Dakota input file.

See `dakota/share/dakota/dakota_experiment_read.in`

7.6.4.5.2 num_experiments

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [num_experiments](#)

Add context to data: number of different experiments

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The number of different experiments. Dakota will expand the total number of residual terms based on the number of calibration terms and the number of experiments. For example, if the number of calibration terms are five scalars, and there are three experiments, the total number of residuals in the least squares formulation will be 15. See [calibration_data](#) or [calibration_data_file](#).

7.6.4.5.3 num_config_variables

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [num_config_variables](#)

Add context to data: number of configuration variables.

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

If there are multiple experiments, there can be different configuration variables (e.g. experimental settings, boundary conditions, etc.) per experiment. See [calibration_data](#) or [calibration_data_file](#).

During calibration, configuration variables are state variables which will be passed to the simulation, and are not treated as calibration parameters.

7.6.4.5.4 experiment_variance_type

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [experiment_variance_type](#)

Add context to data: specify the type of experimental error

Specification

Alias: variance_type

Argument(s): STRINGLIST

Default: none

Description

There are four options for specifying the experimental error (e.g. the measurement error in the data you provide for calibration purposes): 'none' (default), 'scalar', 'diagonal', or 'matrix.'

If the user specifies scalar, they can provide a scalar variance per calibration term. Note that for scalar calibration terms, only 'none' or 'scalar' are options for the measurement variance. However, for field calibration terms, there are two additional options. 'diagonal' allows the user to provide a vector of measurement variances (one for each

term in the calibration field). This vector corresponds to the diagonal of the full covariance matrix of measurement errors. If the user specifies 'matrix', they can provide a full covariance matrix (not just the diagonal terms), where each element (i,j) of the covariance matrix represents the covariance of the measurement error between the i-th and j-th field values.

Usage Tips

Variance information is specified on a per-response group (descriptor), per-experiment basis. Off-diagonal covariance between response groups or between experiments is not supported.

Examples

The figure below shows an observation vector with 5 responses; 2 scalar + 3 field (each field of length > 1). The corresponding covariance matrix has scalar variances σ_1^2, σ_2^2 for each of the scalars s_1, s_2 , diagonal covariance D_3 for field f_3 , scalar covariance σ_4^2 for field f_4 , and full matrix covariance C_5 for field f_5 . In total, Dakota supports block diagonal covariance Σ across the responses, with blocks Σ_i , which could be fully dense within a given field response group. Covariance across the highest-level responses (off-diagonal blocks) is not supported, nor is covariance between experiments.

7.6.4.5.5 scalar_data_file

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [scalar_data_file](#)

Specify a scalar data file to complement field data files (mixed case)

Specification

Alias: none

Argument(s): STRING

Child Keywords:

	Required/Optional Optional (<i>Choose One</i>)	Description of Group Tabular Format (Group 1)	Dakota Keyword custom_annotated	Dakota Keyword Description Selects custom-annotated tabular file format for experiment data
			annotated	Selects annotated tabular file format for experiment data
			freeform	Selects free-form tabular file format for experiment data

Description

When calibrating both scalar and field calibration terms, to associated experimental data, the scalar data may be provided in the file named by `scalar_data_file`. This file follows the same format as: [calibration_data_file](#).

Default Behavior

If `scalar_data_file` is omitted, all calibration data, including for scalar responses, will be read from the generic field `calibration_data` format.

7.6.4.5.5.1 custom_annotated

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [scalar_data_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format for experiment data

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		header	Enable header row in custom-annotated tabular file
	Optional		exp_id	Enable experiment ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file containing experiment data, including configuration variables, observations, and/or observation errors, depending on context. For experiment import, custom-annotated allows user options for whether `header` row and `exp_id` column appear in the tabular file, thus bridging `freeform` and (fully) `annotated`.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. The `custom_annotated` keyword, followed by options can be used to select other formats.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import an experimental data file containing a header row, no leading `exp_id` column, and experiment data in a calibration study

```
responses
...
scalar_data_file 'shock_experiment.dat'
  custom_annotated header
```

Example data file with two measured quantities, three experiments:

```
% velocity stress
18.23 83.21
34.14 93.24
22.41 88.92
```

7.6.4.5.5.2 header

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [scalar_data_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.6.4.5.5.3 exp_id

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)

- [calibration_data](#)
- [scalar_data_file](#)
- [custom_annotated](#)
- [exp_id](#)

Enable experiment ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no exp_id column

Description

See description of parent `custom_annotated`

7.6.4.5.5.4 annotated

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [scalar_data_file](#)
- [annotated](#)

Selects annotated tabular file format for experiment data

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. For experiment data files, each subsequent row contains an experiment ID, followed by data for configuration variables, observations, and/or observation errors, depending on context.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import an annotated experimental data file containing a header row, leading `exp_id` column, and experiment data in a calibration study

```
responses
...
  scalar_data_file 'shock_experiment.dat'
  annotated
```

Example data file with two measured quantities, three experiments:

```
%exp_id    velocity    stress
1          18.23      83.21
2          34.14      93.24
3          22.41      88.92
```

7.6.4.5.5.5 freeform

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [scalar_data_file](#)
- [freeform](#)

Selects free-form tabular file format for experiment data

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. For experiment data files, each row contains data for configuration variables, observations, and/or observation errors, depending on context.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. Specify `freeform` to instead select this format.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import a free-form experimental data file containing raw experiment data in a calibration study

```
responses
...
scalar_data_file 'shock_experiment.dat'
  freeform
```

Example data file with two measured quantities, three experiments:

18.23	83.21
34.14	93.24
22.41	88.92

7.6.4.5.6 interpolate

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data](#)
- [interpolate](#)

Flag to indicate interpolation of simulation values.

Specification

Alias: none

Argument(s): none

Description

If `interpolate` is specified, Dakota will interpolate between the simulation data and the experiment data to calculate the residuals for calibration methods. Specifically, the simulation data is interpolated onto the experimental data points. So, if the simulation data is a field of length 100 with one independent coordinate, and the experiment data is of length 5 with one independent coordinate, the interpolation is done between the 100 (t,f) simulation points (where t is the independent coordinate and f is the simulation field value) onto the five (t_e, f_e) points to obtain the residual differences between the simulation and experiment. See [calibration_data](#).

7.6.4.6 calibration_data_file

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)

Supply scalar calibration data only

Specification

Alias: least_squares_data_file

Argument(s): STRING

Default: none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional (<i>Choose One</i>)	Tabular Format (Group 1)	custom_annotated	Selects custom-annotated tabular file format for experiment data
			annotated	Selects annotated tabular file format for experiment data
			freeform	Selects free-form tabular file format for experiment data
	Optional		num_experiments	Add context to data: number of different experiments
	Optional		num_config_variables	Add context to data: number of configuration variables.
	Optional		experiment_variance_type	Add context to data: specify the type of experimental error

Description

Enables text file import of experimental observations for use in calibration, for scalar responses only, with optional scalar variance information. For more complex data import cases see [calibration_data](#) Dakota will calibrate model variables to best match these data.

Key options include:

- **format:** whether the data file is in `annotated`, `custom_annotated`, or `freeform` format
- **content:** where `num_experiments`, `num_config_variables`, and `experiment_variance_type` indicate which columns appear in the data.

In the most general case, the content of the data file is described by the arguments of three optional parameters.

- `num_experiments` (N_{exp})
 Default: $N_{exp} = 1$
 This indicates that the data represent multiple experiments, where each experiment might be conducted with different values of configuration variables. An experiment can also be thought of as a replicate, where the experiments are run at the same values of the configuration variables.
- `num_config_variables` (N_{cfg})
 Configuration variables specify the values of experimental conditions at which data were collected. The variables in these columns must correspond to state variables in the calibration study. The simulation model will be run at each configuration and compared to the appropriate experiment data.
- `experiment_variance_type` ('none' or 'scalar')
 This indicates if the data file contains variances for measurement error of the experimental data. The default is 'none'.

While some components may be omitted, the most complete version of an annotated calibration data file could include columns corresponding to experiment ID, configuration variables, function value observations, and variances (observation errors), shown here in annotated format:

```
%exp_id | configuration xvars | y data observations | y data variances
1         7.8  7           21.9372  1.8687           0.25  0.04
2         8.6  2           19.0779  4.8976           0.25  0.04
3         8.4  8           38.2758  4.4559           0.25  0.04
4         4.2  1           39.7600  6.4631           0.25  0.04
```

Each row in the file corresponds to an experiment or replicate observation of an experiment to be compared to the model output. This example shows 4 experiments, governed by two configuration variables (one real-valued and one integer-valued), two responses (QOIs), and corresponding observation errors with standard deviation 0.5 and 0.2.

Usage Tips

- The `calibration_data_file` keyword is used when *only* scalar calibration terms are present. If there are field calibration terms, instead use `calibration_data`. For mixed scalar and field calibration terms, one may use the `scalar_data_file` specification, which uses the format described on this page.
- **Attention:** In versions of Dakota prior to 6.14, string-valued configuration variables were specified in data files with 0-based indices into the admissible values. As of Dakota 6.14, strings must be specified by value. For example a string-valued configuration variable for an experimental condition might appear in the file as `low_pressure` vs. `high_pressure`.

Examples

Simple Case: In the simplest case, no data content descriptors are specified:

```
responses
calibration_terms = 2
  descriptors = 'volts' 'amps'
  calibration_data_file = 'circuit.dat'
  annotated
```

And the data file `circuit.dat` must contain only the y^{Data} observations which represent a single experimental observation. In this case, the data file should have $N_{terms} = 2$ columns (for volts, amps) and 1 row, where N_{terms} is the value of `calibration_terms`. The data file is shown here in annotated format:

```
%exp_id | y data observations
1         21.9372  1.8687
```

For each function evaluation, Dakota will run the analysis driver, which must return $N_{terms} = 2$ model responses. Then the residuals are computed as:

$$R_i = y_i^{Model} - y_i^{Data}.$$

These residuals can be weighted using [weights](#).

Multiple experiments: One might specify `num_experiments` N_E indicating that there are multiple experiments. When multiple experiments are present, Dakota will expand the number of residuals for the repeat measurement data and difference with the data accordingly. For example, if the user has $N_E = 4$ experiments in the example above with 2 calibration terms, the input file would contain

```
responses
  calibration_terms = 2
  descriptors = 'volts' 'amps'
  calibration_data_file = 'circuit.dat'
  annotated
  num_experiments = 4
```

And the `calibration_data_file` would need to contain 2 rows (one for each experiment), and each row should contain 2 experimental data values that will be differenced with respect to the appropriate model response:

```
%exp_id | y data observations
1         21.9372  1.8687
2         19.0779  4.8976
3         38.2758  4.4559
4         39.7600  6.4631
```

To summarize, Dakota will calculate the sum of the squared residuals as:

$$f = \sum_{i=1}^{N_E} R_i^2$$

where the residuals now are calculated as:

$$R_i = y_i^{Model}(\theta) - y_i^{Data}.$$

With experimental variances: If information is known about the measurement error and the uncertainty in the measurement, that can be specified by sending the measurement error variance to Dakota. In this case, the keyword `experiment_variance_type` is added, followed by a string of variance types of length one or of length N_{terms} , where N_{terms} is the value of `calibration_terms`. The `experiment_variance_type` for each response can be 'none' or 'scalar'. NOTE: you must specify the same `experiment_variance_type` for all scalar terms. That is, they will all be 'none' or all be 'scalar.'

```
responses
  calibration_terms = 2
  descriptors = 'volts' 'amps'
  calibration_data_file = 'circuit.dat'
  annotated
  experiment_variance_type 'scalar'
```

For each response that has a 'scalar' variance type, each row of the datafile will now have $N_{terms} = 2$ of y data values (volts, amps) followed by $N_{terms} = 2$ columns that specify the measurement error (in units of variance, not standard deviation) for volts, amps. An example with two experiments in annotated format:

```
%exp_id | y data observations | y data variances
1         21.9372  1.8687         0.25  0.04
```

Dakota will run the analysis driver, which must return N_{terms} responses. Then the residuals are computed as:

$$R_i = \frac{y_i^{Model} - y_i^{Data}}{\sqrt{var_i}}$$

for $i = 1 \dots N_{terms}$.

Putting all the options together: Specifying all these options together might look like

```
responses
  calibration_terms = 2
  descriptors = 'volts' 'amps'
  calibration_data_file = 'circuit.dat'
  annotated
  num_experiments = 4
  experiment_variance_type 'scalar'
```

Dakota will expect a data file

%exp_id	configuration	xvars	y data observations	y data variances
1	7.8	7	21.9372 1.8687	0.25 0.04
2	8.6	2	19.0779 4.8976	0.25 0.04
3	8.4	8	38.2758 4.4559	0.25 0.04
4	4.2	1	39.7600 6.4631	0.25 0.04

To compute residuals for each experiment, e.g., `exp_id = 4`, Dakota will

1. Evaluate the computational model at the specified configuration (state variables = [4.2, 1]).
2. Difference the resulting 2 function values with the data [39.7600 volts, 6.4631 amps]
3. Weight by the standard deviation = `sqrt([0.25 0.04])`

See Also

These keywords may also be of interest:

- [calibration_data](#)

7.6.4.6.1 custom_annotated

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [custom_annotated](#)

Selects custom-annotated tabular file format for experiment data

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		<code>header</code>	Enable header row in custom-annotated tabular file
	Optional		<code>exp_id</code>	Enable experiment ID column in custom-annotated tabular file

Description

A custom-annotated tabular file is a whitespace-separated text file containing experiment data, including configuration variables, observations, and/or observation errors, depending on context. For experiment import, custom-annotated allows user options for whether `header` row and `exp_id` column appear in the tabular file, thus bridging `freeform` and (fully) annotated.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. The `custom_annotated` keyword, followed by options can be used to select other formats.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import an experimental data file containing a header row, no leading `exp_id` column, and experiment data in a calibration study

```
responses
...
scalar_data_file 'shock_experiment.dat'
  custom_annotated header
```

Example data file with two measured quantities, three experiments:

```
% velocity stress
18.23 83.21
34.14 93.24
22.41 88.92
```

7.6.4.6.1.1 header

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [custom_annotated](#)
- [header](#)

Enable header row in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no header

Description

See description of parent `custom_annotated`

7.6.4.6.1.2 exp_id

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [custom_annotated](#)
- [exp_id](#)

Enable experiment ID column in custom-annotated tabular file

Specification

Alias: none

Argument(s): none

Default: no exp_id column

Description

See description of parent `custom_annotated`

7.6.4.6.2 annotated

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [annotated](#)

Selects annotated tabular file format for experiment data

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

An annotated tabular file is a whitespace-separated text file with one leading header row of comments/column labels. For experiment data files, each subsequent row contains an experiment ID, followed by data for configuration variables, observations, and/or observation errors, depending on context.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. The `annotated` keyword can be used to explicitly specify this.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import an annotated experimental data file containing a header row, leading `exp_id` column, and experiment data in a calibration study

```
responses
...
scalar_data_file 'shock_experiment.dat'
annotated
```

Example data file with two measured quantities, three experiments:

```
%exp_id    velocity    stress
1          18.23      83.21
2          34.14      93.24
3          22.41      88.92
```

7.6.4.6.3 freeform

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [freeform](#)

Selects free-form tabular file format for experiment data

Topics

This keyword is related to the topics:

- [file_formats](#)

Specification

Alias: none

Argument(s): none

Default: annotated format

Description

A freeform tabular file is whitespace-separated text file with no leading header row and no leading columns. For experiment data files, each row contains data for configuration variables, observations, and/or observation errors, depending on context.

Default Behavior

By default, Dakota imports tabular experiment data files in annotated format. Specify `freeform` to instead select this format.

Usage Tips

- Prior to October 2011, calibration and surrogate data files were in free-form format. They now default to annotated format, though `freeform` remains an option.
- When importing tabular data, a warning will be generated if a specific number of data are expected, but extra is found and an error generated when there is insufficient data.

Examples

Import a free-form experimental data file containing raw experiment data in a calibration study

```
responses
...
scalar_data_file 'shock_experiment.dat'
freeform
```

Example data file with two measured quantities, three experiments:

```
18.23      83.21
34.14      93.24
22.41      88.92
```

7.6.4.6.4 num_experiments

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [num_experiments](#)

Add context to data: number of different experiments

Specification

Alias: none

Argument(s): INTEGER

Default: 1

Description

The number of different experiments. Dakota will expand the total number of residual terms based on the number of calibration terms and the number of experiments. For example, if the number of calibration terms are five scalars, and there are three experiments, the total number of residuals in the least squares formulation will be 15. See [calibration_data](#) or [calibration_data_file](#).

7.6.4.6.5 num_config_variables

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [num_config_variables](#)

Add context to data: number of configuration variables.

Specification

Alias: none

Argument(s): INTEGER

Default: 0

Description

If there are multiple experiments, there can be different configuration variables (e.g. experimental settings, boundary conditions, etc.) per experiment. See [calibration_data](#) or [calibration_data_file](#).

During calibration, configuration variables are state variables which will be passed to the simulation, and are not treated as calibration parameters.

7.6.4.6.6 experiment_variance_type

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [calibration_data_file](#)
- [experiment_variance_type](#)

Add context to data: specify the type of experimental error

Specification

Alias: variance_type

Argument(s): STRINGLIST

Default: none

Description

There are four options for specifying the experimental error (e.g. the measurement error in the data you provide for calibration purposes): 'none' (default), 'scalar', 'diagonal', or 'matrix.'

If the user specifies scalar, they can provide a scalar variance per calibration term. Note that for scalar calibration terms, only 'none' or 'scalar' are options for the measurement variance. However, for field calibration terms, there are two additional options. 'diagonal' allows the user to provide a vector of measurement variances (one for each term in the calibration field). This vector corresponds to the diagonal of the full covariance matrix of measurement errors. If the user specifies 'matrix', they can provide a full covariance matrix (not just the diagonal terms), where each element (i,j) of the covariance matrix represents the covariance of the measurement error between the i-th and j-th field values.

Usage Tips

Variance information is specified on a per-response group (descriptor), per-experiment basis. Off-diagonal covariance between response groups or between experiments is not supported.

Examples

The figure below shows an observation vector with 5 responses; 2 scalar + 3 field (each field of length > 1). The corresponding covariance matrix has scalar variances σ_1^2, σ_2^2 for each of the scalars s_1, s_2 , diagonal covariance D_3 for field f_3 , scalar covariance σ_4^2 for field f_4 , and full matrix covariance C_5 for field f_5 . In total, Dakota supports block diagonal covariance Σ across the responses, with blocks Σ_i , which could be fully dense within a given field response group. Covariance across the highest-level responses (off-diagonal blocks) is not supported, nor is covariance between experiments.

7.6.4.7 simulation_variance

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [simulation_variance](#)

Variance applied to simulation responses

Specification

Alias: none

Argument(s): REALLIST

Default: no variance

Description

The variance that is applied to simulations run by Dakota, i.e. using the `analysis_drivers` command. The user may supply a single variance or a vector of variances of length equal to the number of responses. In both cases, the values provided are treated as scalar variance types. If a single variance is provided, it is applied to all responses produced by the simulation code. If a vector is provided, each variance is applied to the corresponding response output produced by the simulation code.

It is important to note that the the variance defined by this keyword differs from that defined using `experiment_variance_type`. These two commands apply to user-provided calibration data, specified, for example, by `calibration_data` or `calibration_data_file`. However, `simulation_variance` applies to those responses produced by simulation code that is run by Dakota, as described above.

Usage Tips

Currently, this keyword is only in use as part of the algorithm implemented by `experimental_design`. In this algorithm, two models (usually, one high-fidelity and one low-fidelity) are provided to Dakota, each with their own `responses` section of the input script, and each of which is allowed its own `simulation_variance`. The variance specified in the `responses` block belonging to the high-fidelity model is applied to any *new* high-fidelity data that is produced by Dakota running the high-fidelity model. In the `experimental_design` algorithm, low-fidelity model responses are used during the calibration of the model parameters, the calculation of the mutual information, and the calculation of any posterior statistics after the algorithm is complete. The `simulation_variance` is applied to the low-fidelity model responses that are used in the calculation of the mutual information. See the User's and Theory Manuals for more information.

Examples

The example below shows two `responses` blocks, one for the low-fidelity model and one for the high-fidelity model. Both contain `simulation_variance` commands that will apply to the low- and high-fidelity model responses, respectively.

```
responses,
  id_responses = 'low-fidelity'
  calibration_terms = 1
  simulation_variance = 0.5

responses,
  id_responses = 'high-fidelity'
  calibration_terms = 1
  calibration_data_file = 'dakota_bayes_expdesign.dat'
  freeform
```

```

num_config_variables = 1
num_experiments = 1
experiment_variance_type = 'none'
simulation_variance = 1.2

```

7.6.4.8 nonlinear_inequality_constraints

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_inequality_constraints](#)

Group to specify nonlinear inequality constraints

Topics

This keyword is related to the topics:

- [nonlinear_constraints](#)

Specification

Alias: num_nonlinear_inequality_constraints

Argument(s): INTEGER

Default: 0

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		lower_bounds	Specify minimum values
	Optional		upper_bounds	Specify maximum values
	Optional		scale_types	How to scale each nonlinear constraint
	Optional		scales	Characteristic values to scale each nonlinear constraint

Description

Specifies the number of nonlinear inequality constraint functions returned by the interface.

The `lower_bounds` and `upper_bounds` specifications provide the lower and upper bounds for 2-sided nonlinear inequalities of the form

$$g_l \leq g(x) \leq g_u$$

When constraint bounds are not specified, the problem is assumed to have one-sided inequalities bounded above by zero:

$$g(x) \leq 0.0.$$

This provides backwards compatibility with previous Dakota versions.

In a user bounds specification, any upper bound values greater than `+bigRealBoundSize` (`1.e+30`, as defined in `Minimizer`) are treated as `+infinity` and any lower bound values less than `-bigRealBoundSize` are treated as `-infinity`. This feature is commonly used to drop one of the bounds in order to specify a 1-sided constraint (just as the default lower bounds drop out since `-DBL_MAX < -bigRealBoundSize`). The same approach is used for nonexistent linear inequality bounds and for nonexistent design variable bounds.

The `scale_types` and `scales` keywords are related to scaling of $g(x)$. See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.4.8.1 lower_bounds

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_inequality_constraints](#)
- [lower_bounds](#)

Specify minimum values

Specification

Alias: `nonlinear_inequality_lower_bounds`

Argument(s): REALLIST

Default: vector values = `-infinity`

Description

Specify minimum values

7.6.4.8.2 upper_bounds

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_inequality_constraints](#)
- [upper_bounds](#)

Specify maximum values

Specification

Alias: `nonlinear_inequality_upper_bounds`

Argument(s): REALLIST

Default: vector values = `0` .

Description

Specify maximum values

7.6.4.8.3 `scale_types`

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_inequality_constraints](#)
- [scale_types](#)

How to scale each nonlinear constraint

Specification

Alias: `nonlinear_inequality_scale_types`

Argument(s): STRINGLIST

Default: no scaling

Description

Each string in `scale_types` indicates the scaling type for each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

The options are:

- `'value'` - characteristic value by which nonlinear constraint values will be divided. If this is chosen, then `scales` must also be specified; `'value'` is assumed if `scales` are given without `scale_types`
- `'auto'` - automatic scaling based on bounds (inequalities) or targets (equalities)
- `'log'` - logarithmic scaling (can be used together with `scales`, which can be helpfully used to negate values prior to log transformation)

If a single string is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a string must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.4.8.4 `scales`

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_inequality_constraints](#)
- [scales](#)

Characteristic values to scale each nonlinear constraint

Specification

Alias: `nonlinear_inequality_scales`

Argument(s): REALLIST

Default: 1.0 (no scaling)

Description

Each real value in `scales` is a nonzero characteristic value to be used in scaling each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

This keyword is required for `scale_types` of 'value' and optional for 'log'. When specified in conjunction with log, scale values are applied prior to the logarithm, to permit log scaling of strictly negative response functions.

If a single scale value is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a value must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

When `scales` are specified, but not `scale_types`, the scaling type is assumed to be 'value' for this set of nonlinear inequality (equality) constraints.

Use scale values of 1.0 to selectively avoid scaling a subset of response functions.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.4.9 `nonlinear_equality_constraints`

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_equality_constraints](#)

Group to specify nonlinear equality constraints

Topics

This keyword is related to the topics:

- [nonlinear_constraints](#)

Specification

Alias: `num_nonlinear_equality_constraints`

Argument(s): INTEGER

Default: 0

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional	targets	Target values for the nonlinear equality constraint
	Optional	scale_types	How to scale each nonlinear constraint
	Optional	scales	Characteristic values to scale each nonlinear constraint

Description

Specifies the number of nonlinear equality constraint functions returned by the interface.

The `targets` specification provides the targets for nonlinear equalities of the form

$$h(x) = h_t$$

and the defaults for the equality targets enforce a value of 0. for each constraint

$$h(x) = 0.0$$

The `scale_types` and `scales` keywords are related to scaling of $h(x)$. See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.4.9.1 targets

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_equality_constraints](#)
- [targets](#)

Target values for the nonlinear equality constraint

Specification

Alias: `nonlinear_equality_targets`

Argument(s): REALLIST

Default: vector values = 0 .

Description

The `targets` specification provides the targets for nonlinear equalities of the form

$$g(x) = g_t$$

and the defaults for the equality targets enforce a value of 0 . 0 for each constraint:

$$g(x) = 0.0$$

7.6.4.9.2 `scale_types`

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_equality_constraints](#)
- [scale_types](#)

How to scale each nonlinear constraint

Specification

Alias: `nonlinear_equality_scale_types`

Argument(s): STRINGLIST

Default: no scaling

Description

Each string in `scale_types` indicates the scaling type for each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

The options are:

- `'value'` - characteristic value by which nonlinear constraint values will be divided. If this is chosen, then `scales` must also be specified; `'value'` is assumed if `scales` are given without `scale_types`
- `'auto'` - automatic scaling based on bounds (inequalities) or targets (equalities)
- `'log'` - logarithmic scaling (can be used together with `scales`, which can be helpfully used to negate values prior to log transformation)

If a single string is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a string must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.4.9.3 `scales`

- [Keywords Area](#)
- [responses](#)
- [calibration_terms](#)
- [nonlinear_equality_constraints](#)
- [scales](#)

Characteristic values to scale each nonlinear constraint

Specification

Alias: `nonlinear_equality_scales`

Argument(s): REALLIST

Default: 1.0 (no scaling)

Description

Each real value in `scales` is a nonzero characteristic value to be used in scaling each nonlinear inequality (equality) constraint. They only have effect when the associated method specifies `scaling`.

This keyword is required for `scale_types` of 'value' and optional for 'log'. When specified in conjunction with log, scale values are applied prior to the logarithm, to permit log scaling of strictly negative response functions.

If a single scale value is specified it will apply to all of the nonlinear inequality (equality) constraints. Otherwise a value must be specified for each nonlinear inequality (equality) constraint.

Usage Tips:

When `scales` are specified, but not `scale_types`, the scaling type is assumed to be 'value' for this set of nonlinear inequality (equality) constraints.

Use scale values of 1.0 to selectively avoid scaling a subset of response functions.

See the scaling information under specific methods, e.g., `method-*-scaling` for details on how to use this keyword.

7.6.5 response_functions

- [Keywords Area](#)
- [responses](#)
- [response_functions](#)

Generic response type

Specification

Alias: `num_response_functions`

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		scalar_responses	Number of scalar response functions
	Optional		field_responses	Number of field responses functions

Description

A generic response data set is specified using `response_functions`. Each of these functions is simply a response quantity of interest with no special interpretation taken by the method in use.

Whereas objective, constraint, and residual functions have special meanings for optimization and least squares algorithms, the generic response function data set need not have a specific interpretation and the user is free to define whatever functional form is convenient.

Theory

This type of data set is used by uncertainty quantification methods, in which the effect of parameter uncertainty on response functions is quantified, and can also be used in parameter study and design of experiments methods (although these methods are not restricted to this data set), in which the effect of parameter variations on response functions is evaluated.

See Also

These keywords may also be of interest:

- [objective_functions](#)
- [calibration_terms](#)

7.6.5.1 scalar_responses

- [Keywords Area](#)
- [responses](#)
- [response_functions](#)
- [scalar_responses](#)

Number of scalar response functions

Specification

Alias: num_scalar_responses

Argument(s): INTEGER

Description

This keyword describes the number of scalar response functions. It is meant to be used in conjunction with `field_responses`, which describes the number of field response functions. The total number of response functions, both scalar and field, is given by `response_functions`. If only scalar responses functions are specified, it is not necessary to specify the number of scalar terms explicitly: one can simply say `response_functions = 5` and get 5 scalar responses. However, if there are three scalar responses and 2 field responses, then `response_functions = 5` but `scalar_responses = 3` and `field_responses = 2`.

This type of data set is used by uncertainty quantification methods, in which the effect of parameter uncertainty on response functions is quantified, and can also be used in parameter study and design of experiments methods (although these methods are not restricted to this data set), in which the effect of parameter variations on response functions is evaluated.

See Also

These keywords may also be of interest:

- [field_responses](#)

7.6.5.2 field_responses

- [Keywords Area](#)
- [responses](#)
- [response_functions](#)
- [field_responses](#)

Number of field responses functions

Specification

Alias: num_field_responses

Argument(s): INTEGER

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		lengths	Lengths of field responses
	Optional		num_coordinates_per_field	Number of independent coordinates for field responses
	Optional		read_field_coordinates	Add context to data: flag to indicate that field coordinates should be read

Description

This keyword describes the number of field response functions. A field function is a set of related response values collected over a range of independent coordinate values which may or may not be specified by the user. For example, voltage over time would be a field function, where voltage is the `field_objective` and time is the independent coordinate. Similarly, temperature over time and space would be a field response, where the independent coordinates would be both time and spatial coordinates such as (x,y) or (x,y,z), depending on the application. The main difference between scalar responses and field responses is that for field data, we plan to implement methods that take advantage of the correlation or relationship between the field values.

Note that if there is one `field_response`, and it has length 100 (meaning 100 values), then the user's simulation code must return 100 values. Also, if there are both scalar and field responses, the user should specify the number of scalar responses as `scalar_responses`. If there are only field responses, it still is necessary to specify both `response_functions = NN` and `field_responses = NN`, where NN is the number of field responses.

This type of data set is used by uncertainty quantification methods, in which the effect of parameter uncertainty on response functions is quantified, and can also be used in parameter study and design of experiments methods (although these methods are not restricted to this data set), in which the effect of parameter variations on response functions is evaluated. Currently, field response functions will be translated back to scalar responses. So, a field of length 100 will be treated as 100 separate scalar responses. However, in future versions of Dakota, we plan to implement methods which can exploit the nature of field data.

See Also

These keywords may also be of interest:

- [scalar_responses](#)

7.6.5.2.1 lengths

- [Keywords Area](#)
- [responses](#)
- [response_functions](#)
- [field_responses](#)
- [lengths](#)

Lengths of field responses

Specification

Alias: none

Argument(s): INTEGERLIST

Description

This keyword describes the lengths of each field response. It is an integer vector of length `field_responses`. For example, if the `field_responses = 2`, an example would be `lengths = 50 200`, indicating that the first field response has 50 field elements but the second one has 200. The coordinate values (e.g. the independent variables) corresponding to these field responses are read in files labeled `response_descriptor.coords`.

See Also

These keywords may also be of interest:

- [field_responses](#)

7.6.5.2.2 num_coordinates_per_field

- [Keywords Area](#)
- [responses](#)
- [response_functions](#)
- [field_responses](#)
- [num_coordinates_per_field](#)

Number of independent coordinates for field responses

Specification

Alias: none

Argument(s): INTEGERLIST

Description

This keyword describes the number of independent coordinates for each field response. It is an integer vector of length `field_responses`. For example, if the `field_responses = 2`, an example would be `num_coordinates_per_field = 2 1` means that the first field response has two sets of independent coordinates (perhaps x, y locations), but the second response only has one (for example, time where the field response is only dependent upon time). The actual coordinate values (e.g. the independent variables) corresponding to these field responses are defined in a file call `response_descriptor.coords`, where `response_descriptor` is the name of the individual field.

See Also

These keywords may also be of interest:

- [field_responses](#)

7.6.5.2.3 read_field_coordinates

- [Keywords Area](#)
- [responses](#)
- [response_functions](#)
- [field_responses](#)
- [read_field_coordinates](#)

Add context to data: flag to indicate that field coordinates should be read

Specification

Alias: none

Argument(s): none

Description

Field coordinates specify independent variables (e.g. spatial or temporal coordinates) upon which the field depends. For example, the voltage level above might be a function of time, so time is the field coordinate. If the user has field coordinates to read, they need to specify `read_field_coordinates`. The field coordinates will then be read from a file named `response_descriptor.coords`, where `response_descriptor` is the user-provided descriptor for the field response. The number of columns in the `coords` file should be equal to the number of field coordinates.

7.6.6 no_gradients

- [Keywords Area](#)
- [responses](#)
- [no_gradients](#)

Gradients will not be used

Specification

Alias: none

Argument(s): none

Description

The `no_gradients` specification means that gradient information is not needed in the study. Therefore, it will neither be retrieved from the simulation nor computed with finite differences. The `no_gradients` keyword is a complete specification for this case.

See Also

These keywords may also be of interest:

- [numerical_gradients](#)
- [analytic_gradients](#)
- [mixed_gradients](#)

7.6.7 analytic_gradients

- [Keywords Area](#)
- [responses](#)
- [analytic_gradients](#)

Analysis driver will return gradients

Specification

Alias: none

Argument(s): none

Description

The `analytic_gradients` specification means that gradient information is available directly from the simulation (finite differencing is not required). The simulation must return the gradient data in the Dakota format (enclosed in single brackets; see Dakota File Data Formats in the Users Manual [5]) for the case of file transfer of data. The `analytic_gradients` keyword is a complete specification for this case.

See Also

These keywords may also be of interest:

- [numerical_gradients](#)
- [no_gradients](#)
- [mixed_gradients](#)

7.6.8 mixed_gradients

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)

Gradients are needed and will be obtained from a mix of numerical and analytic sources

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required		id_numerical_gradients	Identify which numerical gradient corresponds to which response
	Required		id_analytic_gradients	Identify which analytical gradient corresponds to which response
	Optional		method_source	Specify which finite difference routine is used
	Optional (<i>Choose One</i>)	Gradient Source (Group 1)	dakota	(Default) Use internal Dakota finite differences algorithm
			vendor	Use non-Dakota fd algorithm
	Optional		interval_type	Specify how to compute gradients and Hessians
	Optional (<i>Choose One</i>)	Finite Difference Type (Group 2)	forward	(Default) Use forward differences
			central	Use central differences
	Optional		fd_step_size	Step size used when computing gradients and Hessians

Description

The `mixed_gradients` specification means that some gradient information is available directly from the simulation (analytic) whereas the rest will have to be finite differenced (numerical). This specification allows the user to make use of as much analytic gradient information as is available and then finite difference for the rest.

The `method_source`, `interval_type`, and `fd_gradient_step_size` specifications pertain to those functions listed by the `id_numerical_gradients` list.

Examples

For example, the objective function may be a simple analytic function of the design variables (e.g., weight) whereas the constraints are nonlinear implicit functions of complex analyses (e.g., maximum stress).

See Also

These keywords may also be of interest:

- [numerical_gradients](#)
- [no_gradients](#)
- [analytic_gradients](#)

7.6.8.1 id_numerical_gradients

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [id_numerical_gradients](#)

Identify which numerical gradient corresponds to which response

Topics

This keyword is related to the topics:

- [objective_function_pointer](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The `id_analytic_gradients` list specifies by number the functions which have analytic gradients, and the `id_numerical_gradients` list specifies by number the functions which must use numerical gradients. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_gradients` and `id_numerical_gradients` lists.

See Also

These keywords may also be of interest:

- [id_analytic_gradients](#)

7.6.8.2 `id_analytic_gradients`

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [id_analytic_gradients](#)

Identify which analytical gradient corresponds to which response

Topics

This keyword is related to the topics:

- [objective_function_pointer](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The `id_analytic_gradients` list specifies by number the functions which have analytic gradients, and the `id_numerical_gradients` list specifies by number the functions which must use numerical gradients. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_gradients` and `id_numerical_gradients` lists.

See Also

These keywords may also be of interest:

- [id_numerical_gradients](#)

7.6.8.3 `method_source`

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [method_source](#)

Specify which finite difference routine is used

Specification

Alias: none

Argument(s): none

Default: dakota

Description

The `method_source` setting specifies the source of the finite differencing routine that will be used to compute the numerical gradients:

- `dakota` (default)
- `vendor`

`dakota` denotes Dakota's internal finite differencing algorithm and `vendor` denotes the finite differencing algorithm supplied by the iterator package in use (DOT, CONMIN, NPSOL, NL2SOL, NLSSOL, ROL, and OPT++ each have their own internal finite differencing routines). The `dakota` routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [5]).

However, the `vendor` setting can be desirable in some cases since certain libraries will modify their algorithm when the finite differencing is performed internally. Since the selection of the `dakota` routine hides the use of finite differencing from the optimizers (the optimizers are configured to accept user-supplied gradients, which some algorithms assume to be of analytic accuracy), the potential exists for the `vendor` setting to trigger the use of an algorithm more optimized for the higher expense and/or lower accuracy of finite-differencing. For example, NPSOL uses gradients in its line search when in user-supplied gradient mode (since it assumes they are inexpensive), but uses a value-based line search procedure when internally finite differencing. The use of a value-based line search will often reduce total expense in serial operations. However, in parallel operations, the use of gradients in the NPSOL line search (user-supplied gradient mode) provides excellent load balancing without need to resort to speculative optimization approaches.

In summary, then, the `dakota` routine is preferred for parallel optimization, and the `vendor` routine may be preferred for serial optimization in special cases.

7.6.8.4 dakota

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [dakota](#)

(Default) Use internal Dakota finite differences algorithm

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		ignore_bounds	Do not respect bounds when computing gradients or Hessians

	Optional (<i>Choose One</i>)	Step Scaling (Group 1)	relative	(Default) Scale step size by the parameter value
			absolute	Do not scale step-size
			bounds	Scale step-size by the domain of the parameter

Description

The `dakota` routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [5]).

When the `method_source` is `dakota`, the user may also specify the type of scaling desired when determining the finite difference step size. The choices are `absolute`, `bounds`, and `relative`. For `absolute`, the step size will be applied as is. For `bounds`, it will be scaled by the range of each parameter. For `relative`, it will be scaled by the parameter value.

7.6.8.4.1 ignore_bounds

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [dakota](#)
- [ignore_bounds](#)

Do not respect bounds when computing gradients or Hessians

Specification

Alias: none

Argument(s): none

Default: bounds respected

Description

When Dakota computes gradients or Hessians by finite differences and the variables in question have bounds, it by default chooses finite-differencing steps that keep the variables within their specified bounds. Older versions of Dakota generally ignored bounds when computing finite differences. To restore the older behavior, one can add keyword `ignore_bounds` to the `response` specification when `method_source` `dakota` (or just `dakota`) is also specified.

In forward difference or backward difference computations, honoring bounds is straightforward.

To honor bounds when approximating $\partial f / \partial x_i$, i.e., component i of the gradient of f , by central differences, Dakota chooses two steps h_1 and h_2 with $h_1 \neq h_2$, such that $x + h_1 e_i$ and $x + h_2 e_i$ both satisfy the bounds, and then computes

$$\frac{\partial f}{\partial x_i} \approx \frac{h_2^2(f_1 - f_0) - h_1^2(f_2 - f_0)}{h_1 h_2 (h_2 - h_1)},$$

with $f_0 = f(x)$, $f_1 = f(x + h_1 e_i)$, and $f_2 = f(x + h_2 e_i)$.

7.6.8.4.2 relative

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [dakota](#)
- [relative](#)

(Default) Scale step size by the parameter value

Specification

Alias: none

Argument(s): none

Description

Scale step size by the parameter value

7.6.8.4.3 absolute

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [dakota](#)
- [absolute](#)

Do not scale step-size

Specification

Alias: none

Argument(s): none

Default: relative

Description

Do not scale step-size

7.6.8.4.4 bounds

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [dakota](#)
- [bounds](#)

Scale step-size by the domain of the parameter

Specification

Alias: none

Argument(s): none

Description

Scale step-size by the domain of the parameter

7.6.8.5 vendor

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [vendor](#)

Use non-Dakota fd algorithm

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

7.6.8.6 interval_type

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [interval_type](#)

Specify how to compute gradients and hessians

Specification

Alias: none

Argument(s): none

Default: forward

Description

The `interval_type` setting is used to select between `forward` and `central` differences in the numerical gradient calculations. The `dakota`, `DOT vendor`, and `OPT++ vendor` routines have both forward and central differences available, the `CONMIN`, `NLSOL` and `ROL vendor` routines support forward differences only, and the `NPSOL` and `NLSSOL vendor` routines start with forward differences and automatically switch to central differences as the iteration progresses (the user has no control over this). The following forward difference expression

$$\nabla f(\mathbf{x}) \cong \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

and the following central difference expression

$$\nabla f(\mathbf{x}) \cong \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}$$

are used to estimate the i^{th} component of the gradient vector.

7.6.8.7 forward

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [forward](#)

(Default) Use forward differences

Specification

Alias: none

Argument(s): none

Default: forward

Description

See parent page for usage notes.

7.6.8.8 central

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [central](#)

Use central differences

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

7.6.8.9 `fd_step_size`

- [Keywords Area](#)
- [responses](#)
- [mixed_gradients](#)
- [fd_step_size](#)

Step size used when computing gradients and Hessians

Specification

Alias: `fd_gradient_step_size`

Argument(s): REALLIST

Default: 0.001

Description

`fd_step_size` specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of `absolute`, the differencing interval will be `fd_step_size`.

For Dakota with an interval scaling type of `bounds`, the differencing intervals are computed by multiplying `fd_step_size` with the range of the parameter. For Dakota (with an interval scaling type of `relative`), DOT, CONMIN, and OPT++, the differencing intervals are computed by multiplying the `fd_step_size` with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ all use `.01*fd_step_size` as their minimum absolute differencing interval. With a `fd_step_size = .001`, for example, Dakota, DOT, CONMIN, and OPT++ will use intervals of `.001*current value` with a minimum interval of `1.e-5`. NPSOL and NLSSOL use a different formula for their finite difference intervals: `fd_step_size*(1+|current parameter value|)`. This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.

ROL's finite difference step size can not be controlled via Dakota. Therefore, `fd_step_size` will be ignored when ROL's finite differencing routines are used (vendor FD gradients are specified). ROL's differencing intervals are computed by multiplying the current parameter value with the square root of machine precision.

7.6.9 `numerical_gradients`

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)

Gradients are needed and will be approximated by finite differences

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		method_source	Specify which finite difference routine is used
	Optional (<i>Choose One</i>)	Gradient Source (Group 1)	dakota	(Default) Use internal Dakota finite differences algorithm
			vendor	Use non-Dakota fd algorithm
	Optional		interval_type	Specify how to compute gradients and Hessians
	Optional (<i>Choose One</i>)	Finite Difference Type (Group 2)	forward	(Default) Use forward differences
			central	Use central differences
	Optional		fd_step_size	Step size used when computing gradients and Hessians

Description

The `numerical_gradients` specification means that gradient information is needed and will be computed with finite differences using either the native or one of the vendor finite differencing routines.

See Also

These keywords may also be of interest:

- [no_gradients](#)
- [analytic_gradients](#)
- [mixed_gradients](#)

7.6.9.1 `method_source`

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [method_source](#)

Specify which finite difference routine is used

Specification

Alias: none

Argument(s): none

Default: dakota

Description

The `method_source` setting specifies the source of the finite differencing routine that will be used to compute the numerical gradients:

- `dakota` (default)
- `vendor`

`dakota` denotes Dakota's internal finite differencing algorithm and `vendor` denotes the finite differencing algorithm supplied by the iterator package in use (DOT, CONMIN, NPSOL, NL2SOL, NLSSOL, ROL, and OPT++ each have their own internal finite differencing routines). The `dakota` routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [5]).

However, the `vendor` setting can be desirable in some cases since certain libraries will modify their algorithm when the finite differencing is performed internally. Since the selection of the `dakota` routine hides the use of finite differencing from the optimizers (the optimizers are configured to accept user-supplied gradients, which some algorithms assume to be of analytic accuracy), the potential exists for the `vendor` setting to trigger the use of an algorithm more optimized for the higher expense and/or lower accuracy of finite-differencing. For example, NPSOL uses gradients in its line search when in user-supplied gradient mode (since it assumes they are inexpensive), but uses a value-based line search procedure when internally finite differencing. The use of a value-based line search will often reduce total expense in serial operations. However, in parallel operations, the use of gradients in the NPSOL line search (user-supplied gradient mode) provides excellent load balancing without need to resort to speculative optimization approaches.

In summary, then, the `dakota` routine is preferred for parallel optimization, and the `vendor` routine may be preferred for serial optimization in special cases.

7.6.9.2 dakota

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [dakota](#)

(Default) Use internal Dakota finite differences algorithm

Specification

Alias: none

Argument(s): none

Default: relative

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		ignore_bounds	Do not respect bounds when computing gradients or Hessians
	Optional (<i>Choose One</i>)	Step Scaling (Group 1)	relative	(Default) Scale step size by the parameter value
			absolute	Do not scale step-size
			bounds	Scale step-size by the domain of the parameter

Description

The `dakota` routine is the default since it can execute in parallel and exploit the concurrency in finite difference evaluations (see Exploiting Parallelism in the Users Manual [5]).

When the `method_source` is `dakota`, the user may also specify the type of scaling desired when determining the finite difference step size. The choices are `absolute`, `bounds`, and `relative`. For `absolute`, the step size will be applied as is. For `bounds`, it will be scaled by the range of each parameter. For `relative`, it will be scaled by the parameter value.

7.6.9.2.1 `ignore_bounds`

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [dakota](#)
- [ignore_bounds](#)

Do not respect bounds when computing gradients or Hessians

Specification

Alias: none

Argument(s): none

Default: bounds respected

Description

When Dakota computes gradients or Hessians by finite differences and the variables in question have bounds, it by default chooses finite-differencing steps that keep the variables within their specified bounds. Older versions of Dakota generally ignored bounds when computing finite differences. To restore the older behavior, one can add keyword `ignore_bounds` to the `response` specification when `method_source` `dakota` (or just `dakota`) is also specified.

In forward difference or backward difference computations, honoring bounds is straightforward.

To honor bounds when approximating $\partial f / \partial x_i$, i.e., component i of the gradient of f , by central differences, Dakota chooses two steps h_1 and h_2 with $h_1 \neq h_2$, such that $x + h_1 e_i$ and $x + h_2 e_i$ both satisfy the bounds, and then computes

$$\frac{\partial f}{\partial x_i} \cong \frac{h_2^2(f_1 - f_0) - h_1^2(f_2 - f_0)}{h_1 h_2 (h_2 - h_1)},$$

with $f_0 = f(x)$, $f_1 = f(x + h_1 e_i)$, and $f_2 = f(x + h_2 e_i)$.

7.6.9.2.2 relative

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [dakota](#)
- [relative](#)

(Default) Scale step size by the parameter value

Specification

Alias: none

Argument(s): none

Description

Scale step size by the parameter value

7.6.9.2.3 absolute

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [dakota](#)
- [absolute](#)

Do not scale step-size

Specification

Alias: none

Argument(s): none

Default: relative

Description

Do not scale step-size

7.6.9.2.4 bounds

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [dakota](#)
- [bounds](#)

Scale step-size by the domain of the parameter

Specification

Alias: none

Argument(s): none

Description

Scale step-size by the domain of the parameter

7.6.9.3 vendor

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [vendor](#)

Use non-Dakota fd algorithm

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

7.6.9.4 interval_type

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [interval_type](#)

Specify how to compute gradients and Hessians

Specification

Alias: none

Argument(s): none

Default: forward

Description

The `interval_type` setting is used to select between `forward` and `central` differences in the numerical gradient calculations. The `dakota`, `DOT vendor`, and `OPT++ vendor` routines have both forward and central differences available, the `CONMIN`, `NLSOL` and `ROL vendor` routines support forward differences only, and the `NPSOL` and `NLSSOL vendor` routines start with forward differences and automatically switch to central differences as the iteration progresses (the user has no control over this). The following forward difference expression

$$\nabla f(\mathbf{x}) \cong \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

and the following central difference expression

$$\nabla f(\mathbf{x}) \cong \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}$$

are used to estimate the i^{th} component of the gradient vector.

7.6.9.5 forward

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [forward](#)

(Default) Use forward differences

Specification

Alias: none

Argument(s): none

Default: forward

Description

See parent page for usage notes.

7.6.9.6 central

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [central](#)

Use central differences

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

7.6.9.7 `fd_step_size`

- [Keywords Area](#)
- [responses](#)
- [numerical_gradients](#)
- [fd_step_size](#)

Step size used when computing gradients and Hessians

Specification

Alias: `fd_gradient_step_size`

Argument(s): REALLIST

Default: 0.001

Description

`fd_step_size` specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of `absolute`, the differencing interval will be `fd_step_size`.

For Dakota with an interval scaling type of `bounds`, the differencing intervals are computed by multiplying `fd_step_size` with the range of the parameter. For Dakota (with an interval scaling type of `relative`), DOT, CONMIN, and OPT++, the differencing intervals are computed by multiplying the `fd_step_size` with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ all use `.01*fd_step_size` as their minimum absolute differencing interval. With a `fd_step_size = .001`, for example, Dakota, DOT, CONMIN, and OPT++ will use intervals of `.001*current value` with a minimum interval of `1.e-5`. NPSOL and NLSSOL use a different formula for their finite difference intervals: `fd_step_size*(1+|current parameter value|)`. This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.

ROL's finite difference step size can not be controlled via Dakota. Therefore, `fd_step_size` will be ignored when ROL's finite differencing routines are used (vendor FD gradients are specified). ROL's differencing intervals are computed by multiplying the current parameter value with the square root of machine precision.

7.6.10 `no_hessians`

- [Keywords Area](#)

- [responses](#)
- [no_hessians](#)

Hessians will not be used

Specification

Alias: none

Argument(s): none

Description

The `no_hessians` specification means that the method does not require Dakota to manage the computation of any Hessian information. Therefore, it will neither be retrieved from the simulation nor computed by Dakota. The `no_hessians` keyword is a complete specification for this case. Note that, in some cases, Hessian information may still be being approximated internal to an algorithm (e.g., within a quasi-Newton optimizer such as `optpp_q_newton`); however, Dakota has no direct involvement in this process and the `responses` specification need not include it.

See Also

These keywords may also be of interest:

- [numerical_hessians](#)
- [quasi_hessians](#)
- [analytic_hessians](#)
- [mixed_hessians](#)

7.6.11 numerical_hessians

- [Keywords Area](#)
- [responses](#)
- [numerical_hessians](#)

Hessians are needed and will be approximated by finite differences

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description

	Optional		fd_step_size	Step size used when computing gradients and Hessians
	Optional (<i>Choose One</i>)	Step Scaling (Group 1)	relative	(Default) Scale step size by the parameter value
			absolute	Do not scale step-size
			bounds	Scale step-size by the domain of the parameter
	Optional (<i>Choose One</i>)	Finite Difference Type (Group 2)	forward	(Default) Use forward differences
			central	Use central differences

Description

The `numerical_hessians` specification means that Hessian information is needed and will be computed with finite differences using either first-order gradient differencing (for the cases of `analytic_gradients` or for the functions identified by `id_analytic_gradients` in the case of `mixed_gradients`) or first- or second-order function value differencing (all other gradient specifications). In the former case, the following expression

$$\nabla^2 f(\mathbf{x})_i \cong \frac{\nabla f(\mathbf{x} + h\mathbf{e}_i) - \nabla f(\mathbf{x})}{h}$$

estimates the i^{th} Hessian column, and in the latter case, the following expressions

$$\nabla^2 f(\mathbf{x})_{i,j} \cong \frac{f(\mathbf{x} + h_i\mathbf{e}_i + h_j\mathbf{e}_j) - f(\mathbf{x} + h_i\mathbf{e}_i) - f(\mathbf{x} - h_j\mathbf{e}_j) + f(\mathbf{x})}{h_i h_j}$$

and

$$\nabla^2 f(\mathbf{x})_{i,j} \cong \frac{f(\mathbf{x} + h\mathbf{e}_i + h\mathbf{e}_j) - f(\mathbf{x} + h\mathbf{e}_i - h\mathbf{e}_j) - f(\mathbf{x} - h\mathbf{e}_i + h\mathbf{e}_j) + f(\mathbf{x} - h\mathbf{e}_i - h\mathbf{e}_j)}{4h^2}$$

provide first- and second-order estimates of the ij^{th} Hessian term. Prior to Dakota 5.0, Dakota always used second-order estimates. In Dakota 5.0 and newer, the default is to use first-order estimates (which honor bounds on the variables and require only about a quarter as many function evaluations as do the second-order estimates), but specifying `central` after `numerical_hessians` causes Dakota to use the old second-order estimates, which do not honor bounds. In optimization algorithms that use Hessians, there is little reason to use second-order differences in computing Hessian approximations.

See Also

These keywords may also be of interest:

- [no_hessians](#)
- [quasi_hessians](#)
- [analytic_hessians](#)
- [mixed_hessians](#)

7.6.11.1 `fd_step_size`

- [Keywords Area](#)
- [responses](#)
- [numerical_hessians](#)
- [fd_step_size](#)

Step size used when computing gradients and Hessians

Specification

Alias: `fd_hessian_step_size`

Argument(s): REALLIST

Default: 0.001 (forward), 0.002 (central)

Description

`fd_step_size` specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of `absolute`, the differencing interval will be `fd_step_size`.

For Dakota with an interval scaling type of `bounds`, the differencing intervals are computed by multiplying `fd_step_size` with the range of the parameter. For Dakota (with an interval scaling type of `relative`), DOT, CONMIN, and OPT++, the differencing intervals are computed by multiplying the `fd_step_size` with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ all use `.01*fd_step_size` as their minimum absolute differencing interval. With a `fd_step_size = .001`, for example, Dakota, DOT, CONMIN, and OPT++ will use intervals of `.001*current value` with a minimum interval of `1.e-5`. NPSOL and NLSSOL use a different formula for their finite difference intervals: `fd_step_size*(1+|current parameter value|)`. This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.

ROL's finite difference step size can not be controlled via Dakota. Therefore, `fd_step_size` will be ignored when ROL's finite differencing routines are used (vendor FD gradients are specified). ROL's differencing intervals are computed by multiplying the current parameter value with the square root of machine precision.

7.6.11.2 `relative`

- [Keywords Area](#)
- [responses](#)
- [numerical_hessians](#)
- [relative](#)

(Default) Scale step size by the parameter value

Specification

Alias: none

Argument(s): none

Description

Scale step size by the parameter value

7.6.11.3 absolute

- [Keywords Area](#)
- [responses](#)
- [numerical_hessians](#)
- [absolute](#)

Do not scale step-size

Specification

Alias: none

Argument(s): none

Default: relative

Description

Do not scale step-size

7.6.11.4 bounds

- [Keywords Area](#)
- [responses](#)
- [numerical_hessians](#)
- [bounds](#)

Scale step-size by the domain of the parameter

Specification

Alias: none

Argument(s): none

Description

Scale step-size by the domain of the parameter

7.6.11.5 forward

- [Keywords Area](#)
- [responses](#)
- [numerical_hessians](#)
- [forward](#)

(Default) Use forward differences

Specification

Alias: none

Argument(s): none

Default: forward

Description

See parent page for usage notes.

7.6.11.6 central

- [Keywords Area](#)
- [responses](#)
- [numerical_hessians](#)
- [central](#)

Use central differences

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

7.6.12 quasi_hessians

- [Keywords Area](#)
- [responses](#)
- [quasi_hessians](#)

Hessians are needed and will be approximated by secant updates (BFGS or SR1) from a series of gradient evaluations

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Required (<i>Choose One</i>)	Quasi-Hessian Approximation (Group 1)	bfgs	Use BFGS method to compute quasi-hessians
			sr1	Use the Symmetric Rank 1 update method to compute quasi-Hessians

Description

The `quasi_hessians` specification means that Hessian information is needed and will be approximated using secant updates (sometimes called "quasi-Newton updates", though any algorithm that approximates Newton's method is a quasi-Newton method).

Compared to finite difference numerical Hessians, secant approximations do not expend additional function evaluations in estimating all of the second-order information for every point of interest. Rather, they accumulate approximate curvature information over time using the existing gradient evaluations.

The supported secant approximations include the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update (specified with the keyword `bfgs`) and the Symmetric Rank 1 (SR1) update (specified with the keyword `sr1`).

See Also

These keywords may also be of interest:

- [no_hessians](#)
- [numerical_hessians](#)
- [analytic_hessians](#)
- [mixed_hessians](#)

7.6.12.1 bfgs

- [Keywords Area](#)
- [responses](#)
- [quasi_hessians](#)
- [bfgs](#)

Use BFGS method to compute quasi-hessians

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		damped	Numerical safeguarding for BFGS updates

Description

Broyden-Fletcher-Goldfarb-Shanno (BFGS) update will be used to compute quasi-Hessians.

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

where B_k is the k^{th} approximation to the Hessian, $s_k = x_{k+1} - x_k$ is the step and $y_k = \nabla f_{k+1} - \nabla f_k$ is the corresponding yield in the gradients.

Notes

- Initial scaling of $\frac{y_k^T y_k}{y_k^T s_k} I$ is used for B_0 prior to the first update.
- Numerical safeguarding is used to protect against numerically small denominators within the updates.
- This safeguarding skips the update if $|y_k^T s_k| < 10^{-6} s_k^T B_k s_k$
- Additional safeguarding can be added using the `damped` option, which utilizes an alternative damped BFGS update when the curvature condition $y_k^T s_k > 0$ is nearly violated.

7.6.12.1.1 damped

- [Keywords Area](#)
- [responses](#)
- [quasi_hessians](#)
- [bfgs](#)
- [damped](#)

Numerical safeguarding for BFGS updates

Specification

Alias: none

Argument(s): none

Default: undamped BFGS

Description

See parent page.

7.6.12.2 sr1

- [Keywords Area](#)
- [responses](#)
- [quasi_hessians](#)
- [sr1](#)

Use the Symmetric Rank 1 update method to compute quasi-Hessians

Specification

Alias: none

Argument(s): none

Description

The Symmetric Rank 1 (SR1) update (specified with the keyword `sr1`) will be used to compute quasi-Hessians.

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$$

where B_k is the k^{th} approximation to the Hessian, $s_k = x_{k+1} - x_k$ is the step and $y_k = \nabla f_{k+1} - \nabla f_k$ is the corresponding yield in the gradients.

Notes

- Initial scaling of $\frac{y_k^T y_k}{y_k^T s_k} I$ is used for B_0 prior to the first update.
- Numerical safeguarding is used to protect against numerically small denominators within the updates.
- This safeguarding skips the update if $|(y_k - B_k s_k)^T s_k| < 10^{-6} \|s_k\|_2 \|y_k - B_k s_k\|_2$

7.6.13 analytic_hessians

- [Keywords Area](#)
- [responses](#)
- [analytic_hessians](#)

Hessians are needed and are available directly from the analysis driver

Specification

Alias: none

Argument(s): none

Description

The `analytic_hessians` specification means that Hessian information is available directly from the simulation. The simulation must return the Hessian data in the Dakota format (enclosed in double brackets; see Dakota File Data Formats in Users Manual [5]) for the case of file transfer of data. The `analytic_hessians` keyword is a complete specification for this case.

See Also

These keywords may also be of interest:

- [no_hessians](#)
- [numerical_hessians](#)
- [quasi_hessians](#)
- [mixed_hessians](#)

7.6.14 mixed_hessians

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)

Hessians are needed and will be obtained from a mix of numerical, analytic, and "quasi" sources

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		id_numerical_hessians	Identify which numerical-Hessian corresponds to which response
	Optional (<i>Choose One</i>)	Step Scaling (Group 1)	relative	(Default) Scale step size by the parameter value
			absolute	Do not scale step-size
			bounds	Scale step-size by the domain of the parameter
	Optional (<i>Choose One</i>)	Finite Difference Type (Group 2)	forward	(Default) Use forward differences
			central	Use central differences
	Optional		id_quasi_hessians	Identify which quasi-Hessian corresponds to which response

	Optional	id_analytic_hessians	Identify which analytical Hessian corresponds to which response
--	-----------------	--------------------------------------	---

Description

The `mixed_hessians` specification means that some Hessian information is available directly from the simulation (analytic) whereas the rest will have to be estimated by finite differences (numerical) or approximated by secant updating. As for mixed gradients, this specification allows the user to make use of as much analytic information as is available and then estimate/approximate the rest.

The `id_analytic_hessians` list specifies by number the functions which have analytic Hessians, and the `id_numerical_hessians` and `id_quasi_hessians` lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_hessians`, `id_numerical_hessians`, and `id_quasi_hessians` lists.

The `fd_hessian_step_size` and `bfgs`, `damped_bfgs`, or `srl` secant update selections are as described previously in [responses](#) and pertain to those functions listed by the `id_numerical_hessians` and `id_quasi_hessians` lists.

See Also

These keywords may also be of interest:

- [no_hessians](#)
- [numerical_hessians](#)
- [quasi_hessians](#)
- [analytic_hessians](#)

7.6.14.1 id_numerical_hessians

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [id_numerical_hessians](#)

Identify which numerical-Hessian corresponds to which response

Topics

This keyword is related to the topics:

- [objective_function_pointer](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		fd_step_size	Step size used when computing gradients and Hessians

Description

The `id_analytic_hessians` list specifies by number the functions which have analytic Hessians, and the `id_numerical_hessians` and `id_quasi_hessians` lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_hessians`, `id_numerical_hessians`, and `id_quasi_hessians` lists.

See Also

These keywords may also be of interest:

- [id_analytic_hessians](#)
- [id_quasi_hessians](#)

7.6.14.1.1 `fd_step_size`

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [id_numerical_hessians](#)
- [fd_step_size](#)

Step size used when computing gradients and Hessians

Specification

Alias: `fd_hessian_step_size`

Argument(s): REALLIST

Default: 0.001 (forward), 0.002 (central)

Description

`fd_step_size` specifies the relative finite difference step size to be used in the computations. Either a single value may be entered for use with all parameters, or a list of step sizes may be entered, one for each parameter.

The latter option of a list of step sizes is only valid for use with the Dakota finite differencing routine. For Dakota with an interval scaling type of `absolute`, the differencing interval will be `fd_step_size`.

For Dakota with an interval scaling type of `bounds`, the differencing intervals are computed by multiplying `fd_step_size` with the range of the parameter. For Dakota (with an interval scaling type of `relative`), `DOT`, `CONMIN`, and `OPT++`, the differencing intervals are computed by multiplying the `fd_step_size` with the current parameter value. In this case, a minimum absolute differencing interval is needed when the current parameter

value is close to zero. This prevents finite difference intervals for the parameter which are too small to distinguish differences in the response quantities being computed. Dakota, DOT, CONMIN, and OPT++ all use `.01*fd_step_size` as their minimum absolute differencing interval. With a `fd_step_size = .001`, for example, Dakota, DOT, CONMIN, and OPT++ will use intervals of `.001*current value` with a minimum interval of `1.e-5`. NPSOL and NLSSOL use a different formula for their finite difference intervals: `fd_step_size*(1+|current parameter value|)`. This definition has the advantage of eliminating the need for a minimum absolute differencing interval since the interval no longer goes to zero as the current parameter value goes to zero.

ROL's finite difference step size can not be controlled via Dakota. Therefore, `fd_step_size` will be ignored when ROL's finite differencing routines are used (vendor FD gradients are specified). ROL's differencing intervals are computed by multiplying the current parameter value with the square root of machine precision.

7.6.14.2 relative

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [relative](#)

(Default) Scale step size by the parameter value

Specification

Alias: none

Argument(s): none

Description

Scale step size by the parameter value

7.6.14.3 absolute

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [absolute](#)

Do not scale step-size

Specification

Alias: none

Argument(s): none

Default: relative

Description

Do not scale step-size

7.6.14.4 bounds

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [bounds](#)

Scale step-size by the domain of the parameter

Specification

Alias: none

Argument(s): none

Description

Scale step-size by the domain of the parameter

7.6.14.5 forward

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [forward](#)

(Default) Use forward differences

Specification

Alias: none

Argument(s): none

Default: forward

Description

See parent page for usage notes.

7.6.14.6 central

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [central](#)

Use central differences

Specification

Alias: none

Argument(s): none

Description

See parent page for usage notes.

7.6.14.7 id_quasi_hessians

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [id_quasi_hessians](#)

Identify which quasi-Hessian corresponds to which response

Topics

This keyword is related to the topics:

- [objective_function_pointer](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Child Keywords:

	Required/Optional Required (<i>Choose One</i>)	Description of Group Quasi-Hessian Approximation (Group 1)	Dakota Keyword bfgs	Dakota Keyword Description Use BFGS method to compute quasi-hessians
			sr1	Use the Symmetric Rank 1 update method to compute quasi-Hessians

Description

The `id_analytic_hessians` list specifies by number the functions which have analytic Hessians, and the `id_numerical_hessians` and `id_quasi_hessians` lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_hessians`, `id_numerical_hessians`, and `id_quasi_hessians` lists.

See Also

These keywords may also be of interest:

- [id_numerical_hessians](#)
- [id_analytic_hessians](#)

7.6.14.7.1 bfgs

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [id_quasi_hessians](#)
- [bfgs](#)

Use BFGS method to compute quasi-hessians

Specification

Alias: none

Argument(s): none

Child Keywords:

	Required/Optional	Description of Group	Dakota Keyword	Dakota Keyword Description
	Optional		damped	Numerical safeguarding for BFGS updates

Description

Broyden-Fletcher-Goldfarb-Shanno (BFGS) update will be used to compute quasi-Hessians.

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

where B_k is the k^{th} approximation to the Hessian, $s_k = x_{k+1} - x_k$ is the step and $y_k = \nabla f_{k+1} - \nabla f_k$ is the corresponding yield in the gradients.

Notes

- Initial scaling of $\frac{y_k^T y_k}{y_k^T s_k} I$ is used for B_0 prior to the first update.
- Numerical safeguarding is used to protect against numerically small denominators within the updates.
- This safeguarding skips the update if $|y_k^T s_k| < 10^{-6} s_k^T B_k s_k$
- Additional safeguarding can be added using the `damped` option, which utilizes an alternative damped BFGS update when the curvature condition $y_k^T s_k > 0$ is nearly violated.

7.6.14.7.1.1 damped

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [id_quasi_hessians](#)
- [bfgs](#)
- [damped](#)

Numerical safeguarding for BFGS updates

Specification

Alias: none

Argument(s): none

Default: undamped BFGS

Description

See parent page.

7.6.14.7.2 sr1

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [id_quasi_hessians](#)
- [sr1](#)

Use the Symmetric Rank 1 update method to compute quasi-Hessians

Specification

Alias: none

Argument(s): none

Description

The Symmetric Rank 1 (SR1) update (specified with the keyword `sr1`) will be used to compute quasi-Hessians.

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$$

where B_k is the k^{th} approximation to the Hessian, $s_k = x_{k+1} - x_k$ is the step and $y_k = \nabla f_{k+1} - \nabla f_k$ is the corresponding yield in the gradients.

Notes

- Initial scaling of $\frac{y_k^T y_k}{y_k^T s_k} I$ is used for B_0 prior to the first update.
- Numerical safeguarding is used to protect against numerically small denominators within the updates.
- This safeguarding skips the update if $|(y_k - B_k s_k)^T s_k| < 10^{-6} \|s_k\|_2 \|y_k - B_k s_k\|_2$

7.6.14.8 id_analytic_hessians

- [Keywords Area](#)
- [responses](#)
- [mixed_hessians](#)
- [id_analytic_hessians](#)

Identify which analytical Hessian corresponds to which response

Topics

This keyword is related to the topics:

- [objective_function_pointer](#)

Specification

Alias: none

Argument(s): INTEGERLIST

Description

The `id_analytic_hessians` list specifies by number the functions which have analytic Hessians, and the `id_numerical_hessians` and `id_quasi_hessians` lists specify by number the functions which must use numerical Hessians and secant Hessian updates, respectively. Each function identifier, from 1 through the total number of functions, must appear once and only once within the union of the `id_analytic_hessians`, `id_numerical_hessians`, and `id_quasi_hessians` lists.

See Also

These keywords may also be of interest:

- [id_numerical_hessians](#)
- [id_quasi_hessians](#)

7.6.15 metadata

- [Keywords Area](#)
- [responses](#)
- [metadata](#)

(Experimental) Labels for floating point response metadata

Specification

Alias: none

Argument(s): STRINGLIST

Default: no metadata

Description

Specify the labels (field names) for floating point-valued metadata responses to be returned from the interface.

Dakota's emerging metadata responses are distinct from the response quantities of interest explored by a Dakota [method](#). Some, such as simulation cost might be used by a method (such as `method-multifidelity_sampling` and related methods) to make algorithmic decisions. Other metadata might just be tabulated for user convenience.

Usage Tips

In a [results_file](#), values and labels for metadata should follow any function values, gradients, or Hessians returned by the simulation.

Examples

A responses specification where the Dakota method will be applied to quantities of interest stress and displacement, while metadata simulation cost in minutes and number of dead elements will be tabulated for each evaluation:

```
responses
  descriptors 'stress' 'displacement'
  response_functions 2
  no_gradients
  no_hessians
  metadata 'simulation_cost_minutes' 'number_dead_elements'
```

See Also

These keywords may also be of interest:

- [cost_recovery_metadata](#)

Bibliography

- [1] Computational investigations of low-discrepancy sequences. *ACM Transactions on Mathematical Software*, 23(2):266–294, 1997. [3147](#), [3272](#)
- [2] M.A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., S. Le Digabel, and C. Tribes. The NOMAD project. Software available at <http://www.gerad.ca/nomad>. [540](#)
- [3] B. M. Adams, W. J. Bohnhoff, R. A. Canfield, K. R. Dalbey, M. S. Ebeida, J. P. Eddy, M. S. Eldred, G. Geraci, R. W. Hooper, P. D. Hough, K. T. Hu, J. D. Jakeman, K. Carson, M. Khalil, K. A. Maupin, J. A. Monschke, E. M. Ridgway, A. A. Rushdi, D. T. Seidl, J. A. Stephens, L. P. Swiler, A. Tran, D. M. Vigil, T. M. Wildey, J. G. Winokur, and (with Menhorn, F. and Zeng, X.). *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.15 Developers Manual*. Sandia National Laboratories, Albuquerque, NM, November 2021. Available online from <http://dakota.sandia.gov/documentation.html>. [1](#)
- [4] B. M. Adams, W. J. Bohnhoff, R. A. Canfield, K. R. Dalbey, M. S. Ebeida, J. P. Eddy, M. S. Eldred, G. Geraci, R. W. Hooper, P. D. Hough, K. T. Hu, J. D. Jakeman, K. Carson, M. Khalil, K. A. Maupin, J. A. Monschke, E. M. Ridgway, A. A. Rushdi, D. T. Seidl, J. A. Stephens, L. P. Swiler, A. Tran, D. M. Vigil, T. M. Wildey, J. G. Winokur, and (with Menhorn, F. and Zeng, X.). *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.15 Reference Manual*. Sandia National Laboratories, Albuquerque, NM, November 2021. Available online from <http://dakota.sandia.gov/documentation.html>. [165](#), [167](#)
- [5] B. M. Adams, W. J. Bohnhoff, K. R. Dalbey, M. S. Ebeida, J. P. Eddy, M. S. Eldred, R. W. Hooper, P. D. Hough, K. T. Hu, J. D. Jakeman, M. Khalil, K. A. Maupin, J. A. Monschke, E. M. Ridgway, A. A. Rushdi, D. T. Seidl, J. A. Stephens, L. P. Swiler, and J. G. Winokur. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.15 users manual. Technical Report SAND2021-14253, Sandia National Laboratories, Albuquerque, NM, November 2021. Available online from <http://dakota.sandia.gov/documentation.html>. [1](#), [8](#), [9](#), [34](#), [124](#), [156](#), [163](#), [174](#), [176](#), [177](#), [180](#), [189](#), [259](#), [261](#), [263](#), [268](#), [269](#), [271](#), [274](#), [275](#), [276](#), [277](#), [283](#), [285](#), [291](#), [292](#), [294](#), [302](#), [307](#), [308](#), [309](#), [398](#), [1135](#), [1485](#), [3180](#), [3183](#), [3289](#), [3555](#), [3557](#), [3559](#), [3858](#), [3869](#), [3875](#), [3876](#), [3880](#), [3886](#), [3888](#), [3889](#), [3890](#), [3892](#), [3895](#), [3897](#), [3907](#), [3911](#), [3974](#), [3978](#), [3979](#), [3985](#), [3986](#), [3998](#)
- [6] G. Anderson and P. Anderson. *The UNIX C Shell Field Guide*. Prentice-Hall, Englewood Cliffs, NJ, 1986. [4](#)
- [7] J. S. Arora. *Introduction to Optimum Design*. McGraw-Hill, New York, 1989. [174](#)
- [8] C. Audet, S. Le Digabel, and C. Tribes. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD, 2009. [539](#)
- [9] J.-P. Berrut and L. N. Trefethen. Barycentric lagrange interpolation. *SIAM Review*, 46(3):501–517, 2004. [170](#)
- [10] B. J. Bichon, M. S. Eldred, L. P. Swiler, S. Mahadevan, and J. M. McFarland. Multimodal reliability assessment for complex engineering applications using efficient global optimization. In *Proceedings of the 48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference (9th AIAA Non-Deterministic Approaches Conference)*, number AIAA-2007-1946, Honolulu, HI, April 2007. [3220](#)
- [11] B. J. Bichon, M. S. Eldred, L. P. Swiler, S. Mahadevan, and J. M. McFarland. Efficient global reliability analysis for nonlinear implicit performance functions. *AIAA Journal*, 46(10):2459–2468, 2008. [3220](#)
- [12] K. Breitung. Asymptotic approximation for multinormal integrals. *J. Eng. Mech., ASCE*, 110(3):357–366, 1984. [3183](#)

- [13] R. H. Byrd, R. B. Schnabel, and G. A. Schultz. Parallel quasi-newton methods for unconstrained optimization. *Mathematical Programming*, 42:273–306, 1988. [335](#), [343](#), [352](#), [361](#), [370](#), [378](#), [387](#), [403](#), [413](#), [430](#), [446](#), [462](#), [478](#), [494](#), [770](#)
- [14] K. J. Chang, R. T. Haftka, G. L. Giles, and P.-J. Kao. Sensitivity-based scaling for approximating structural response. *J. Aircraft*, 30:283–288, 1993. [3513](#), [3539](#)
- [15] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. MPS-SIAM Series on Optimization, SIAM-MPS, Philadelphia, 2000. [311](#), [312](#), [313](#), [314](#)
- [16] K. R. Dalbey, M. S. Eldred, J. D. Geraci, G. Jakeman, K. A. Maupin, J. A. Monschke, D. T. Seidl, L. P. Swiler, A. Tran, and (with Menhorn, F. and Zeng, X.). Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.15 theory manual. Technical Report SAND2021-14254, Sandia National Laboratories, Albuquerque, NM, November 2021. Available online from <http://dakota.sandia.gov/documentation.html>. [1135](#), [1261](#), [1378](#), [1485](#), [1560](#), [2409](#), [2414](#), [2451](#), [3050](#), [3076](#)
- [17] A. Der Kiureghian and P. L. Liu. Structural reliability under incomplete information. *J. Eng. Mech., ASCE*, 112(EM-1):85–104, 1986. [168](#)
- [18] Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41:637–676, 1999. [3138](#)
- [19] J. E. Eddy and K. Lewis. Effective generation of pareto sets using genetic programming. In *Proceedings of ASME Design Engineering Technical Conference*, 2001. [190](#)
- [20] A. S. El-Bakry, R. A. Tapia, T. Tsuchiya, and Y. Zhang. On the formulation and theory of the newton interior-point method for nonlinear programming. *Journal of Optimization Theory and Applications*, 89:507–541, 1996. [440](#), [456](#), [472](#), [488](#)
- [21] M. S. Eldred, H. Agarwal, V. M. Perez, S. F. Wojtkiewicz, Jr., and J. E. Renaud. Investigation of reliability method formulations in DAKOTA/UQ. *Structure & Infrastructure Engineering: Maintenance, Management, Life-Cycle Design & Performance*, 3(3):199–213, 2007. [166](#)
- [22] M. S. Eldred and B. J. Bichon. Second-order reliability formulations in DAKOTA/UQ. In *Proceedings of the 47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, number AIAA-2006-1828, Newport, RI, May 1–4 2006. [3183](#)
- [23] M. S. Eldred and D. M. Dunlavy. Formulations for surrogate-based optimization with data fit, multifidelity, and reduced-order models. In *Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, number AIAA-2006-7117, Portsmouth, VA, September 6–8 2006. [302](#), [307](#), [308](#), [309](#)
- [24] M. S. Eldred, A. A. Giunta, and S. S. Collis. Second-order corrections for surrogate-based optimization with model hierarchies. In *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, NY, Aug. 30–Sept. 1, 2004. AIAA Paper 2004-4457. [3183](#), [3512](#), [3513](#), [3539](#), [3540](#)
- [25] G. M. Fadel, M. F. Riley, and J.-F. M. Barthelemy. Two point exponential approximation method for structural optimization. *Structural Optimization*, 2(2):117–124, 1990. [3531](#)
- [26] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed. Duxbury Press/Brooks/Cole Publishing Co., Pacific Grove, CA, 2003. For small examples, e.g., at most 300 variables, a student version of AMPL suffices; see <http://www.ampl.com/DOWNLOADS>. [3894](#)
- [27] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, March 1991. [3386](#)
- [28] J. Gablonsky. Direct version 2.0 userguide technical report. Technical Report CRSC-TR01-08, North Carolina State University, Center for Research in Scientific Computation, Raleigh, NC, 2001. [782](#)
- [29] D. M. Gay. Hooking your solver to AMPL. Technical Report Technical Report 97-4-06, Bell Laboratories, Murray Hill, NJ, 1997. Available online as <http://www.ampl.com/REFS/HOOKING/index.html> and <http://www.ampl.com/REFS/hooks2.pdf> and <http://www.ampl.com/REFS/hooks2.ps.gz>. [3858](#), [3894](#)
- [30] D. M. Gay. Specifying and reading program input with NIDR. Technical Report SAND2008-2261P, Sandia National Laboratories, 2008. Available as <http://dakota.sandia.gov/papers/nidr08.pdf>. [29](#)

- [31] R. Ghanem and J. R. Red-Horse. Propagation of probabilistic uncertainty in complex physical systems using a stochastic finite element technique. *Physica D*, 133:137–144, 1999. [166](#), [170](#)
- [32] R. G. Ghanem and P. D. Spanos. *Stochastic Finite Elements: A Spectral Approach*. Springer-Verlag, New York, 1991. [166](#), [170](#)
- [33] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for NPSOL (Version 4.0): A Fortran package for nonlinear programming. Technical Report TR SOL-86-2, System Optimization Laboratory, Stanford University, Stanford, CA, 1986. [191](#), [399](#), [410](#)
- [34] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, San Diego, CA, 1981. [27](#), [174](#)
- [35] A. A. Giunta. Use of data sampling, surrogate models, and numerical optimization in engineering design. In *Proc. 40th AIAA Aerospace Science Meeting and Exhibit*, number AIAA-2002-0538, Reno, NV, January 2002. [3514](#), [3540](#)
- [36] A. A. Giunta, L. P. Swiler, S. L. Brown, M. S. Eldred, M. D. Richards, and E. C. Cyr. The surpack software library for surrogate modeling of sparse, irregularly spaced multidimensional data. In *Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, number AIAA-2006-7049, Portsmouth, VA, 2006. [3330](#)
- [37] Alex A. Gorodetsky, Gianluca Geraci, Michael S. Eldred, and John D. Jakeman. A generalized approximate control variate framework for multifidelity uncertainty quantification. *Journal of Computational Physics*, 408:109257, 2020. [1800](#)
- [38] G. A. Gray and T. G. Kolda. Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. *ACM Transactions on Mathematical Software*, 32(3):485–507, September 2006. [190](#), [522](#)
- [39] M. Gunburger and J. Burkardt. Uniformity measures for point samples in hypercubes, 2004. Available on John Burkardt's web site: <http://www.csit.fsu.edu/~burkardt/>. [3132](#), [3142](#), [3275](#)
- [40] Heikki Haario, Marko Laine, Antonietta Mira, and Eero Saksman. DRAM: Efficient adaptive MCMC. *Statistics and Computing*, 16:339–354, 2006. [2404](#), [2444](#), [3043](#)
- [41] R. T. Haftka. Combining global and local approximations. *AIAA Journal*, 29(9):1523–1525, 1991. [3512](#), [3539](#)
- [42] R. T. Haftka and Z. Gurdal. *Elements of Structural Optimization*. Kluwer, Boston, 1992. [174](#)
- [43] A. Haldar and S. Mahadevan. *Probability, Reliability, and Statistical Methods in Engineering Design*. Wiley, New York, 2000. [166](#), [3180](#), [3804](#)
- [44] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960. [3147](#), [3272](#)
- [45] J. H. Halton and G. B. Smith. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7:701–702, 1964. [3147](#), [3272](#)
- [46] W. E. Hart, A. A. Giunta, A. G. Salinger, and B. G. van Bloemen Waanders. An overview of the adaptive pattern search algorithm and its application to engineering optimization problems. In *Proceedings of the McMaster Optimization Conference: Theory and Applications*, Hamilton, Ontario, Canada, 2001. [658](#), [664](#)
- [47] J. C. Helton and F. J. Davis. Sampling-based methods for uncertainty and sensitivity analysis. Technical Report SAND99-2240, Sandia National Laboratories, Albuquerque, NM, 2000. [165](#), [168](#)
- [48] J. C. Helton and W. L. Oberkampf. Special issue of reliability engineering and system safety: Issue on alternative representations of epistemic uncertainty, Jul–Sep 2004. [172](#)
- [49] D. Higdon, J. Gattiker, B. Williams, and M. Rightley. Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, 103(482):570–583, 2008. [2423](#)
- [50] N. J. Higham. The numerical stability of barycentric lagrange interpolation. *IMA Journal of Numerical Analysis*, 24(4):547–556, 2004. [170](#)
- [51] M. Hohenbichler and R. Rackwitz. Improvement of second-order reliability estimates by importance sampling. *J. Eng. Mech., ASCE*, 114(12):2195–2199, 1988. [3183](#)

- [52] H.P. Hong. Simple approximations for improving second-order reliability estimates. *J. Eng. Mech., ASCE*, 125(5):592–595, 1999. [3183](#)
- [53] R. L. Iman and W. J. Conover. A distribution-free approach to inducing rank correlation among input variables. *Communications in Statistics: Simulation and Computation*, B11(3):311–334, 1982. [3804](#)
- [54] R. L. Iman and M. J Shortencarier. A Fortran 77 program and user's guide for the generation of latin hypercube samples for use with computer models. Technical Report NUREG/CR-3624, SAND83-2365, Sandia National Laboratories, Albuquerque, NM, 1984. [165](#), [168](#)
- [55] D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998. [803](#)
- [56] M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society*, 63:425–464, 2001. [2423](#)
- [57] W. A. Klimke. *Uncertainty Modeling using Fuzzy Arithmetic and Sparse Grids*. PhD thesis, Universität Stuttgart, Stuttgart, Germany, 2005. [170](#)
- [58] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004. [3067](#), [3076](#), [3077](#)
- [59] R. M. Lewis and S. N. Nash. A multigrid approach to the optimization of systems governed by differential equations. In *Proceedings of the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, number AIAA-2000-4890, Long Beach, CA, Sep 2000. [3512](#), [3513](#), [3539](#)
- [60] J. M. McFarland. *Uncertainty Analysis for Computer Simulations through Validation and Calibration*. PhD thesis, Vanderbilt University, Nashville, Tennessee, 2008. available for download at <http://etd.library.vanderbilt.edu/ETD-db/available/etd-03282008-125137/>. [818](#), [1987](#), [2010](#), [2057](#), [2082](#), [2121](#), [2466](#), [2756](#), [3228](#), [3355](#)
- [61] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. [165](#), [168](#)
- [62] J. C. Meza, R. A. Oliva, P. D. Hough, and P. J. Williams. OPT++: an object oriented toolkit for nonlinear optimization. *ACM Transactions on Mathematical Software*, 33(2), 2007. [191](#)
- [63] J. More and D. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3):286–307, 1994. [437](#), [438](#), [439](#), [453](#), [454](#), [455](#), [469](#), [470](#), [471](#), [485](#), [486](#), [487](#)
- [64] M. D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991. [3151](#)
- [65] R. H. Myers and D. C. Montgomery. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. John Wiley & Sons, Inc., New York, 1995. [3457](#)
- [66] A. Nealen. A short-as-possible introduction to the least squares, weighted least squares, and moving least squares methods for scattered data approximation and interpolation. Technical report, Discrete Geometric Modeling Group, Technische Universitaet, Berlin, Germany, 2004. [3399](#)
- [67] J. Nocedal and Wright S. J. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 1999. [174](#)
- [68] W.T. Nutt and G.B. Wallis. Evaluation of nuclear safety from the outputs of computer codes in the presence of uncertainties. *Reliability Engineering and System Safety*, 83:57–77, 2004. [1658](#)
- [69] W. .L. Oberkampf and J. C. Helton. Evidence theory for engineering applications. Technical Report SAND2003-3559P, Sandia National Laboratories, Albuquerque, NM, 2003. [172](#)
- [70] M. J. L. Orr. Introduction to radial basis function networks. Technical report, University of Edinburgh, Edinburgh, Scotland, 1996. [3443](#)
- [71] V. M. Pérez, J. E. Renaud, and L. T. Watson. An interior-point sequential approximation optimization methodology. *Structural and Multidisciplinary Optimization*, 27(5):360–370, July 2004. [315](#), [316](#)

- [72] T. D. Plantenga. HOPSPACK 2.0 user manual. Technical Report SAND2009-6265, Sandia National Laboratories, 2009. [190](#)
- [73] E. Prudencio and S. H. Cheung. Parallel adaptive multilevel sampling algorithms for the bayesian analysis of mathematical models. *International Journal for Uncertainty Quantification*, 2:215–237, 2012. [2407](#)
- [74] D. G. Robinson and C. Atcity. Comparison of quasi- and pseudo-monte carlo sampling for reliability and uncertainty analysis. In *Proceedings of the AIAA Probabilistic Methods Conference*, number AIAA99-1589, St. Louis, MO, 1999. [3279](#)
- [75] M. Rosenblatt. Remarks on a multivariate transformation. *Annals of Mathematical Statistics*, 23(3):470–472, 1952. [168](#)
- [76] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. John Wiley & Sons, 2004. [871](#), [948](#), [1031](#), [1110](#), [1238](#), [1356](#), [1461](#), [1538](#), [1618](#), [1652](#), [3134](#), [3144](#), [3151](#), [3276](#)
- [77] K. Schittkowski. NLPQLP: A fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search – user’s guide. Technical report, Department of Mathematics, University of Bayreuth, Bayreuth, Germany, 2004. [191](#)
- [78] G. D. Sjaardema. APREPRO: An algebraic preprocessor for parameterizing finite element analyses. Technical Report SAND92-2291, Sandia National Laboratories, Albuquerque, NM, 1992. [3869](#), [3880](#)
- [79] R. Srinivasan. *Importance Sampling*. Springer-Verlag, 2002. [1833](#)
- [80] A. Stroud. *Approximate Calculation of Multiple Integrals*. Prentice Hall, 1971. [1151](#), [2147](#), [2492](#), [2782](#)
- [81] L. P. Swiler and N. J. West. Importance sampling: Promises and limitations. In *Proceedings of the 12th AIAA Non-Deterministic Approaches Conference*, number AIAA-2010-2850, 2010. [1833](#)
- [82] G. Tang, L. P. Swiler, and M. S. Eldred. Using stochastic expansion methods in evidence theory for mixed aleatory-epistemic uncertainty quantification. In *Proceedings of the 51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference (12th AIAA Non-Deterministic Approaches conference)*, Orlando, FL, April 12-15, 2010. AIAA Paper 2010-XXXX. [166](#)
- [83] R. A. Tapia and M. Argaez. Global convergence of a primal-dual interior-point newton method for nonlinear programming using a modified augmented lagrangian function. (In Preparation). [441](#), [457](#), [473](#), [489](#)
- [84] C. H. Tong. The PSUADE software library. Web site, 2005. http://www.llnl.gov/CASC/uncertainty_quantification/#psuade. [192](#)
- [85] R. J. Vanderbei and D. F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–259, 1999. [441](#), [457](#), [473](#), [489](#)
- [86] G. N. Vanderplaats. CONMIN – a FORTRAN program for constrained function minimization. Technical Report TM X-62282, NASA, 1973. See also Addendum to Technical Memorandum, 1978. [188](#)
- [87] G. N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design: With Applications*. McGraw-Hill, New York, 1984. [174](#)
- [88] Vanderplaats Research and Development, Inc., Colorado Springs, CO. *DOT Users Manual, Version 4.20*, 1995. [189](#), [331](#), [340](#), [349](#), [357](#), [366](#)
- [89] J. A. Vrugt, C. J. F. ter Braak, C. G. H. Diks, B. A. Robinson, J. M. Hyman, and D. Higdon. Accelerating Markov chain Monte Carlo simulation by self-adaptive differential evolution with randomized subspace sampling. *International Journal of Nonlinear Scientific Numerical Simulation*, 10(3), 2009. [2111](#), [2748](#)
- [90] V. G. Weirs, J. R. Kamm, L. P. Swiler, M. Ratto, S. Tarantola, B. M. Adams, W. J. Rider, and M. S. Eldred. Sensitivity analysis techniques applied to a system of hyperbolic conservation laws. *Reliability Engineering and System Safety*, 107:157–170, 2012. [871](#), [948](#), [1031](#), [1110](#), [1238](#), [1356](#), [1461](#), [1538](#), [1618](#), [1652](#), [3134](#), [3144](#), [3276](#)
- [91] S. S. Wilks. Determination of sample sizes for setting tolerance limits. *Ann. Math. Stat.*, 12(1):91–96, 1941. [1658](#)

-
- [92] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997. [442](#), [458](#), [474](#), [490](#)
- [93] G. D. Wyss and K. H. Jorgensen. A user's guide to LHS: Sandia's Latin hypercube sampling software. Technical Report SAND98-0210, Sandia National Laboratories, Albuquerque, NM, 1998. [3725](#), [3730](#), [3770](#), [3793](#)
- [94] D. Xiu. Numerical integration formulas of degree two. *Applied Numerical Mathematics*, 58:1515–1520, 2008. [1151](#), [2147](#), [2492](#), [2782](#)
- [95] S. Xu and R. V. Grandhi. Effective two-point function approximation for design optimization. *AIAA J.*, 36(12):2269–2275, 1998. [3183](#), [3530](#), [3531](#)
- [96] D. C. Zimmerman. Genetic algorithms for navigating expensive and complex design spaces, September 1996. Final Report for Sandia National Laboratories contract AO-7736 CA 02. [3429](#)