# PSUADE Reference Manual (Version 2.0)

Charles Tong
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551-0808
September, 2022

# 1 Introduction

PSUADE (Problem Solving environment for Uncertainty Analysis and Design Exploration) is a software package for performing various uncertainty quantification (UQ) activities such as uncertainty analysis (UA), sensitivity analysis (SA), parameter study, numerical optimization under uncertainty, statistical inference, optimal experimental design, and etc. PSUADE comprises three major components: a suite of sampling methods, a job execution environment, and a collection of analysis tools. PSUADE can be launched in three modes:

- In 'Batch' mode, users prepare a PSUADE input file (e.g. psuade.in), and then launch PSUADE with:

```
[Linux] psuade psuade.in
```

- In 'Command Line' mode, users can interact directly with PSUADE by launching it without argument:

```
[Linux] psuade
```

- In 'Library' mode, PSUADE is used a library of functions that are called directly by user programs.

This document focuses on explaining the 'Command Line' mode with its commands, and also the 'Library' mode. How to use the 'Batch' mode is described in the user short manual.

# 2 PSUADE Command Line Interpreter

Available commands in the 'Command Line' mode can be divided roughly into the following categories:

- Commands for read/write from/to files
- Commands for basic statistics
- Commands for input parameter screening
- Commands for response surface analysis
- Commands for RS-based uncertainty/sensitivity analysis
- Commands for optimization/statistical inference
- Commands for optimal experimental designs
- Commands for high-dimensional inference with KPCA

- Commands for creating visualization plots
- Commands for setting up PSUADE runs
- Commands for sample editing
- Other miscellaneous and advanced commands

Also, to provide enhanced capabilities and flexibility, PSUADE interacts with users at different levels, often via various expert modes and diagnostics levels.

## 2.1 Commands for Read/Write from/to Files

This section describes commands that performs read or write operations between PSUADE's workspace and files with different formats. The file format supported are PSUADE data format, a standard format with one sample point per line, comma-separated value (or CSV) format, Excel format, and etc. The PSUADE data format is the most exotic, capturing information such as sample points, input names and ranges, input distributions, sampling methods, runtime information, and many others.

**clear** :

This command cleans up the PSUADE workspace (allocated C++ arrays) so the previously-loaded sample will be erased.

**load** <**file**> : where <file> is a PSUADE data file.

This command loads a sample file in PSUADE data format to the PSUADE workspace. When loaded successfully, the sample data (inputs/outputs) are ready to be analyzed or manipulated. Users can, for example, use sinfo or listall to examine what have been loaded.

**loadmore** <**file**> : where <file> is a PSUADE data file.

This command is used to append another sample (in PSUADE data format) to the end of the sample previously loaded by the load command (Note: calling a load again will erase the current sample in PSUADE's workspace.) In order for this command to be successful, the appended sample must have the same number of inputs and outputs as the previously-loaded sample.

**write** <**file**> : where <file> is a PSUADE data file.

This command writes a sample from the PSUADE workspace to a file in the PSUADE data format. An option is provided to save the sample with just one output or all outputs.

**nwrite** <**file**> : where <file> is a data file.

This command writes only the unevaluated sample points (those with sample state = 0 or those with UNDEFINDED outputs) to the specified file.

**iread** <**file**> : where <file> is a data file.

This command reads a set of sample inputs from a file with a special format (the first line has two integers specifying the sample size and number of inputs, and each of the subsequent lines should have the sample number and the input values for each sample point). An example is: (with 4 sample points and 2 input variables):

```
4 2
1 0.0 0.0
2 1.0 0.0
3 0.0 1.0
4 1.0 1.0
```

**iwrite** <**file**> : where <file> is a data file.

This command writes ONLY the sample inputs of the loaded sample to a file in the format that can be read by the iread command (See above).

**owrite** <**file**> : where <file> is a data file.

This command writes ONLY the outputs of the loaded sample to a file which will have the first line specifying the sample size and number of outputs, and subsequent lines wih each line specifying the sample number and the output values for each sample point.

**read_std** <**file**> : where <file> is a data file.

This command is similar to the load command except that this command reads the sample data in a different format (called standard format, which has the first line specifying the sample size, the number of inputs, and the number of outputs; and subsequent lines specifying sample input and output values). An example is (with 4 sample points, 2 input and 1 output variables):

```
4 2 1
0.0 0.0 0.0
1.0 0.0 1.0
0.0 1.0 1.0
1.0 1.0 2.0
```

**write_std** <**file**> : where <file> is a data file.

This command writes the sample in PSUADE's workspace to the specified file in standard format that can be read by the read_std command (See above).

**read_xls** <**file**> : where <file> is a data file.

This command is similar to read_std except that it adds the sample number at the beginning of each line. An example is:

```
4 2 1
1 0.0 0.0 0.0
2 1.0 0.0 1.0
3 0.0 1.0 1.0
4 1.0 1.0 2.0
```

**write_xls <file>** : where <file> is a data file.

This command writes the sample in PSUADE's workspace to a file in a format that can be read by the read_xls command.

**read_csv <file>** : where <file> is a special CSV file.

This command reads in a sample in the CSV format. The data file has the following format (where the first line may optionally be used to specify which columns are inputs and which are outputs; the second line may optionally be used to specify the input and output names; and subsequent lines are the sample input and output values separated by commas as the delimiter. If the first line is not specified, users will be asked for how many values in each line are inputs and how many are outputs):

```
input,input,input,output,output (optional line)
I1, I2, I3, O1, O2 (optional line)
0.0,0.0,0.0,0.0,0.0
1.0.0.0.1.0.1.0,2.0
0.0.1.0.1.0,2.0,3.0
1.0.1.0.2.0,3.0,4.0
```

**write_csv <file>** : where <file> is a data file.

This command writes the sample in PSUADE's workspace to a file in the CSV format that can be read by the read_csv command.

**write_matlab <file>** : where <file> is a data file.

This command writes the sample in the PSUADE workspace to a file in Matlab format. The sample input and output matrices will be named $X$ and $Y$, respectively, in the Matlab file. Users can append more Matlab codes to this file for further analysis or plotting.

**oupdate <file>** : where <file> is a data file in PSUADE format.

This command updates the sample outputs in PSUADE's workspace from the file <file>. For each sample point in the file, PSUADE will search its workspace to find the same sample point and if found, will modify the workspace data with the new outputs from the file (inputs are not touched). This command requires the number of outputs in the data file and in the PSUADE workspace be the same.

**iadd <file>** : where <file> is a standard PSUADE data file.

Suppose you have a PSUADE data file, and you would like to add a few more inputs to it. You will have to prepare another PSUADE data file (<file>) which has the same sample size. For example, if you have a sample with 2 inputs loaded in PSUADE's workspace and another sample with 3 inputs in another data file, then after issuing this command, the total number of inputs in PSUADE's workspace will be 5 (the new inputs will be appended to the end of the input list). This command is useful if you want to, for example, add a few dummy input variables in your sample data file. Note that no sample outputs will be modified.

**oadd <file>** : where <file> is a data file in PSUADE format.

This command is analogous to iadd. This command only checks that both samples have the same sample size. It does not check that the two sets of sample inputs are indeed the same (so beware).

**ireplace <file>** : where <file> is a data file.

This command is similar to iadd except that instead of adding more inputs, it replaces ONE of the current inputs (to be selected by users) by another input in <file> (NOTE: this file should contain values for only one input parameter). The format of the data file is (the sample size in the file, 4 in this example, should match the sample size in the workspace):

```
PSUADE_BEGIN (optional line)
4
0.0
1.0
2.0
3.0
PSUADE_END (optional line)
```

**oreplace <file>** : where <file> is a data file.

This command is similar to ireplace except that it replaces ALL sample outputs with the data in the given file. The format of the data file is (so the outputs of the sample points in the workspace will be replaced with the 2 outputs in this file):

```
PSUADE_BEGIN (optional line)
4 2
0.0 1.0
1.0 4.0
2.0 3.0
3.0 2.0
PSUADE_END (optional line)
```

This command may be useful if you run simulations on the sample outside of PSUADE, and you want to update your PSUADE sample file with the actual simulation outputs. The sample size of this file should match the size of the same in the workspace.

**ssplit** : split the sample in PSUADE's workspace into two subsets.

This command splits the sample in the workspace into two subsets. It will ask for the sample size of the first subset. The two subsets are stored in psuadeSample1 and psuadeSample2, respectively. A few splitting options are available. NOTE: the sample in PSUADE's workspace will have been destroyed after this command.

## 2.2 Commands for Basic Statistics

This section describes commands for various uncertainty and sensitivity analyses directly on the raw sample resident in PSUADE's workspace, as well as other commands for generating samples from distributions.

**ua** : perform uncertainty analysis.

This command performs uncertainty analaysis directly on the sample in the workspace. It displays basic statistics information such as mean, standard deviation, skewness and kurtosis. This command also creates Matlab plots for visualization.

**sem** : perform standard error of mean analysis.

This command computes the mean of the loaded sample and also the standard error of mean based on bootstrapping. This standard error of mean is compared against the one computed from the computed variance (that is, $\sigma^2/\sqrt{(N)}$ where $N$ is the sample size.

**ca** : perform correlation analysis.

This command performs correlation analaysis directly on the sample in the workspace. It displays correlation information such as Pearson, Spearman and Kendall coefficients. Pearson coefficients are useful sensitivity indicators when the input-output relationship is more or less LINEAR. For nonlinear but monotonic functions, the Spearman coefficients are better indicators. For general nonliniear functions, global sensitivity analysis is recommended.

**me** : perform variance-based main effect analysis.

This command performs main effect analaysis directly on the sample in the workspace. It displays first-order Sobol' indices for the sample inputs. Ideally, this command works best with replicated Latin hypercube samples. It will still work otherwise, but the results may be less less accurate when the sample size is small. This command is intended for computationally inexpensive functions that you can evaluate many times (hundreds of thousands) quickly and when you cannot create a good response surface

for the sample (e.g. due to nonsmooth function). Otherwise, response surface-based methods are recommended. Turn on the analysis expert mode for more advanced analysis.

**ie** : perform variance-based two-way interaction effect analysis.

This command performs interaction effect analaysis directly on the sample in the workspace. It displays second-order Sobol' indices for the sample input pairs. Ideally, this command works best with replicated orthogonal array samples. It will still work otherwise, but the results will be less accurate when the sample size is small. This command is intended for computationally inexpensive functions that you can evaluate many times (hundreds of thousands) quickly and when you cannot create a good response surface for the sample (e.g. due to nonsmooth function). Otherwise, response surface-based methods are recommended.

**tsi** : perform variance-based total sensitivity analysis.

This command performs total effect analaysis directly on the sample in the workspace. It requires a VERY LARGE sample to obtain accurate results and currently it only works with low input dimensions (no more than 10). This command is intended for computationally inexpensive functions that you can evaluate many times (hundreds of thousands) quickly and when you cannot create a good response surface for the sample (e.g. due to nonsmooth function). Otherwise, response surface-based methods are recommended.

**sobol** : perform Sobol' first and total order sensitivity analysis on a Sobol' sample.

This command uses another popular method (beside me and 'tsi') to perform first and total-order sensitivity analysis on the sample in the workspace. This analysis is valid only for a specific sampling design: the Sobol' sampling design (that is, first create a Sobol' design, then run simulations, and finally load the sample and run this command).

**fast** : perform the FAST total-order sensitivity analysis.

This command uses the Fourier Amplitude Sensitivity Test (FAST) method on the sample in the workspace to perform total-order sensitivity analysis. It works only on the FAST sampling designs.

**anova** : perform analysis of variation.

This command performs 'ANOVA' (analysis of variation) on a response surface built from the sample in the workspace.

**1stest** : perform one-sample tests.

This command performs one-sample tests on a sample (not the sample in the workspace, but one to be specified by users) such as the Chi-squared test or the distribution tests (search for mean and standard deviation for the sample).

**2stest** : perform two-sample tests.

This command performs two-sample tests on two samples (not the sample in the workspace) such as the Student T-test (do two normally distributed populations differ?), the Mann-Whitney test (do the two sample have the same probability distribution?), and Kolgomorov Smirnov test (does the sample come from a population with a specific distribution with a given mean and standard deviation?). You will have to enter 2 sample files in the format requested by this command.

**gendist** : create a one-input sample using a user-specified probability distribution.

This command creates a ONE-INPUT sample based on the selected probability distribution. The result sample is written to a file 'sample1D' which has the first line specifying the sample size and subsequent lines specifying the sample input values. For multiple inputs, use the gensample command.

**pdfconvert** : convert a sample based on its probability distribution.

This command converts the inputs of the sample in the workspace (assumed uniformly distribued to begin with) based on their probability distributions (defined in the INPUT section of the data file itself) and replaces the sample inputs with the transformed values. The transformed sample resides in the workspace and it can be written to a PSUADE data file using, e.g., write.

**rand_draw** : draw a sample from the loaded sample with replacement.

This command draws from the loaded sample a bootstrapped sample of user-specified size with replacement. The results will be written to a PSUADE data file with a user-given file name.

**rand_draw2** : draw a sample from two samples with two different sets of inputs.

When initiated, this command requests from users two sample files in PSUADE data format and then randomly draws a new new sample of size $N$ (user-specified) with each sample point a concatenation of one randomly-drawn sample point from each file. The results will be written to another PSUADE data file with a user-given file name.

**rand_drawb** : draw a sample from the loaded sample with replacement (but in blocks).

This command draws from the sample in the workspace a bootstrapped sample of user-specified size with replacement. The difference between this command and rand_draw is that the sample in the workspace is first divided into blocks, then a new sample is created by randomly drawing blocks of contiguous sample points from the loaded sample. This command is useful for nondeterministic simulations when the loaded sample has blocks of contiguous sample points whereby each block has identical sample points but the outputs are different due to stochasticity in the simulation results.

**gensample** : generate a sample based on input parameter information in the INPUT block of the loaded sample.

This command is useful when working with, for example, the rsuab command, which requires a user to enter a sample file for propagating uncertainty through the response surface. This command uses the input parameter information prescribed in the INPUT section of the loaded sample and creates a sample of size specified by users that follows the prescribed probability distributions.

**cdf_lookup** : look up cumulative probability given a probability distribution and the value of a random variable.

This command is useful for computing the probability mass within an interval. For example, given $X1$ and $X2 > X1$, one can compute the probability between $X1$ and $X2$ by looking up probabilities for both (say, $P1$ and $P2$) and then calculating the difference ($P2 - P1$).

**icdf_lookup** : look up the value of a random variable given a probability distribution.

This command is the inverse of cdf_lookup and normally called 'probit' in the statistics community.

**kde** : perform kernel density estimation on a sample.

This command uses a kernel density estimation algorithm to approximate the probability distribution of the loaded sample and then create a large sample from this distribution for histogramming into bins. The histogram can be visualized using the Matlab (or Scilab) file called 'matlabkde.m' (or 'scilabkde.sci').

**kde2** : create probability distribution histograms in chunks for replicated samples.

This command is similar to kde except that it first divides the sample into chunks and then apply kde to each chunk. This command is useful for heteroskechastic models when the sample can be divided into chunks whereby each chunk has the same input values but the outputs are different due to stochastic simulations, and the stochastic distribution for each chunk is to be characterized. The results are written to a file called 'kde2_results'.

## 2.3 Commands for Input Parameter Screening

This section describes commands for various methods for screening of input parameters that are 'NOT CORRELATED' (correlated inputs may use methods such as principal component analysis on the covariance matrix to perform dimension reduction). As such, these commands reduce dimension by eliminating the 'unimportant' ones. In summary, local sensitivity analysis is the cheapest (requiring $m + 1$ simulations for $m$ inputs) but it is suitable primarily for linear models (that is, model output is a linear combination of the inputs), the Morris method is the cheapest (requiring $r(m + 1)$ simulations where $r \sim 4 - 10$ for general nonlinear models) and it can handle hundreds to thousands of inputs, but it is considered a qualitative screening method because the magnitudes of the Morris metric do not precisely quantify the sensitivity of the inputs. Other methods (approximate response surface

methods, the delta test, and the sum-of-trees) are also qualitative method and they can be applied to models with moderate number of inputs (10's) and large sample sizes.

**lsa** : perform local sensitivity screening analysis.

This command displays screening information based on a gradient analysis on the loaded sample. This command works only with local sensitivity analysis (LSA) sampling designs, and the result may not be valid for highly nonlinear models, which require the Morris and/or global sensitivity analysis methods.

**moat** : perform Morris screening analysis.

This command performs the Morris one-at-a-time (MOAT) screening (which is a type of coarse parameter screening methods where the numerical results are qualitative or not very precise) on the sample in the workspace. It displays screening information such as modified mean and standard deviations of the Morris gradients. Analysis results are also saved in a Matlab/Scilab file for visualization. This command works only with MOAT samples and the results are generally valid for nonlinear models (although Type I or Type II error may occur).

**moatmo** : perform Morris analysis on multiple outputs.

This command is similar to the moat command except that it analyzes all outputs and displays the results together in a multi-output heat map in Matlab.

**ff** : perform fractional factorial screening analysis.

This command performs a special type of screening on the loaded sample. This method assumes that the function is linear with respect to each input variable with possible two-way interactions. This command works only with fractional or full factorial samples.

**mars_sa** : perform MARS screening analysis.

This command operates on the sample in the workspace, and it displays screening information based on the MARS importance ranking. The screening information will NOT be displayed on some platform where MARS outputs are suppressed or when there is an error in MARS analysis. This method represents a 'COARSE' (that is, the numerical results are not precise) sensitivity analysis with no linearity assumption on the model.

**gp_sa** : perform Gaussian process screening analysis.

This command operates on the sample in the workspace, and it displays screening information based on importance ranking from Gaussian process or Kriging analysis (the relative magnitudes of the hyperparameters). This is another screening method (besides mars_sa) for general nonlinear models.

**delta_test** : perform Delta test for screening input parameters.

This command operates on the sample in the workspace (and the sample should be a random or quasi-random sample and not structured sample such as factorial). This test is useful for dimension reduction, that is, to identify inputs that are important in driving output variability. It works best for large samples ($ll$1000) and can be used for sample with inputs up to about 100. This test involves numerical optimization and so it can be computationally intensive for large sample size and large number of inputs. During the optimization, the current selections and their corresponding objective values are displayed. Then results of the best parititoning (into important and unimportant input groups) will be displayed.

**eta_test** : perform Eta test for screening input parameters.

This command operates on the sample in the workspace (and the sample should be a random or quasi-random sample and not structured sample such as factorial). The Eta test is a variant of the delta test for dimension reduction, that is, to identify inputs that are important in driving output variability. It works best for large samples ($ll$1000) with number of inputs up to maybe about 50.

**sot_sa** : perform sum-of-trees screening analysis.

This command operates on the loaded sample (should be a space-filling sample). It displays screening information based on a sum-of-trees analysis (the input dimension that receives the most bisections is the most important input). This method is good for large samples with moderate to high input dimensions.

## 2.4 Commands for Response Surface Analysis

This section describes commands for creating and analyzing response surfaces.

**set_rstype** : change the response surface type in the loaded sample.

This command changes the 'rstype' variable in the loaded sample. For some analyses such as rssobol1, PSUADE does not prompt users for the response surface type, but instead looks up 'rstype' from the loaded sample. Thus, this command is provided to allow users run these type of analyses with different response surfaces. Any write operation performed after this command will store the new response surface type to a file.

**svmfind** : find the best parameter setting for support vector machine (SVM) radial basis function (RBF) response surface.

This command uses the loaded sample to perform hyperparameter study with the SVM method. SVM has two parameters if RBF is selected. This command tunes these parameters to obtain the best training errors. Subsequently, these parameters values can be entered in rscheck or other response surface plot commands (when the response surface expert mode is turned on).

**rscheck/rsvalidate/rsfit** : check the quality of different response surfaces on the loaded sample.

This command creates a response surface based on the method selected by users. At the end it computes training errors (and store them in the file 'RSFA_training_err.m' if the analysis mode is turned on). Optionally, cross validation can also be performed. To do that, you will need to select how many groups (that is, the $k$, typically 10, in $k$-fold cross validation). At the end of cross validation, some summary statistics will be displayed on the screen. Specifically, examine especially the magnitudes of the root-mean-squared errors, the maximum errors, and the maximum errors with respect to the amplitude of the output variations to determine the goodness of the fit. In addition, PSUADE creates the file 'RSFA_CV_err.m', a Matlab file useful for visualizing the distribution of cross validation errors. When this file is run, two plots will be displayed. Ideally, the error distribution on the left plot should center around zero with very small spread. Also, on the right plot, ideally all sample points lie on the diagonal line (meaning the predicted and true sample data match well). For further detailed analysis, the 'morePlots' variable inside 'RSFA_CV_err.m' should be set to 1 before launching Matlab, which displays two additional figures (Figure 2 and 3). Figure 2 displays the normalized errors against the sample numbers (that is, point-wise errors divided by the true values provided that the latter are not 0). Figure 3 displays the CV errors against each input parameter. Figure 2 is useful in pinpointing which sample points might have caused large response surface errors and are thus potential outliers. Figure 3 is useful in finding out which inputs and their ranges that might have caused large response surface errors. This command is useful for finding out which response surface method gives the best fit (maybe there is none, in which case you may want to add more sample points, or inspect your sample for outliers).

**rstest** : this corresponds to a suite of tests for the goodness of a response surface created from the loaded sample.

This command builds a response surface for the loaded sample and may ask for a test set (in PSUADE data format) to compute prediction errors. There are four different types of response surface test that check the quality of the response surface:

- rstest_ts: against the training test,
- rstest_hs: against a user-provided hold-out test set,
- rstest_cv: against each other via the k-fold cross validation, and
- rstest_gt: against each other in a more challenging sense by sample bisection method (generalization test).
- rstest_sd: compute prediction uncertainties in the entire parameter space via a large space-filling sample (and plot histogram of standard deviations). As opposed to other rstest's, this command does not need the test sample outputs.

**rscreate** : create a response surface from the loaded sample and keep it in the workspace.

This command builds a response surface for the loaded sample. Once this command has been completed, the response surface is readily available for prediction via the rseval or rseval_m command. This command is useful if you desire to construct response surface once (which may be computationally expensive) and then evaluate the response surface on multiple samples (multiple rseval calls).

**rseval** : evaluate a sample from the response surface constructed from the loaded sample using rscreate.

This command should be used after recreate has been called. You can evaluate a single sample point from the local sample vector (created by ivec_create) or multiple sample points from a file. The file may have PSUADE_BEGIN as its first line. The second line should have the number of inputs, and subsequent lines should have the input values (each line begins with the input number followed by the input value).

**rseval_m** : use PSUADE as a response surface interpolator that can be called by an external client program.

This command is similar to *rseval* except that it uses a different protocol. Similar to *rseval*, this command assumes that a sample has already been loaded to the workspace and rscreate has been called. After this command is launched, it waits for users to put the new sample point in a file (called 'rsevalDataIn.x' starting with $x = 1$), evaluates the sample points, writes the results to a file (called 'rsevalDataOut.x'), and wait for another point or a termination signal (the presence of the 'psComplete' file in the current directory). This command thus allows another program to request multiple response surface evaluations from PSUADE without directly calling PSUADE and only have to construct the response surface once. Note: this file-based server-client protocol is not robust (from the perspective of computer science), but it should work.

**rsevaluate** : evaluate a response surface constructed from the loaded sample on a new sample to be entered by users.

This command is equivalent to calling rscreate and rseval together, and the results will be written into a file called 'rsevaluate.out'. The difference is that after this command has been called and the sample evaluated, the response surface will be destroyed, while once rscreate has been called, the response surface persists until PSUADE is terminated.

**rsevaluate2** : evaluate a response surface constructed from the loaded sample on a new sample to be entered by users.

This command is different from rsevaluate in that users can select some of the input parameters to be 'uncertain parameters' for estimating prediction uncertainty (that is, prediction uncertainty is not from the response surface as in rsevaluate but from the uncertain parameters in the function). This command is equivalent to the mcmc_predict command and is useful for computing prediction uncertainties based on some MCMC posterior sample.

**rs_splot** : create scatter plots from a response surface.

This command is similar to splot except that, instead of creating scatter plots directly from the loaded sample, this command first constructs a response surface from the loaded sample, then generates a large random sample, evaluates the large sample with the response surface, and finally uses the evaluated large sample for creating the scatter plots. The scatter plots are stored in the Matlab file called 'matlabrssp.m'. This command is useful for visualizing trends when the original sample is small.

**rsvol** : calculate volume of the constrained space.

This command first builds a response surface from the loaded sample, then creates a large sample, and finally tabulates the number of sample points that satisfy some given constraint. The percentage of 'feasible' sample points is the percentage of the feasible space in the given parameter space.

**rsint** : perform numerical integration.

This command uses the loaded sample to build a response surface and then computes the integral under the response curve within the parameter ranges.

**rstgen** : create a sample for response surface test.

Response surfaces usually give poor interpolations at the corners or edges or faces of the parameter space due to 'extrapolation'. This command creates a sample at the corners of the parameter space, and is intended to be used with rstest_hs for validating the response model at the corners. Depending on the number of inputs, one can select either the full factorial or fractional factorial sampling design.

**ivec_create/vcreate** : create a vector (register) in the workspace for use in rseval.

This command is intended to be used with the rseval command, which may take the evaluation point from a file or the internal register. If the internal register is used, a user needs first to create the register using ivec_create (or vcreate), then use ivec_modify (or vmodify) to set the sample input values, and finally use rs_eval to evaluate the sample point in the register.

**ivec_modify/vmodify** : modify the content of the sample vector (an internal vector register for storing a single sample point) in PSUADE's workspace.

This command is to be used after an internal sample vector has been created using ivec_create (or vcreate) to modify the content of the sample point register.

**ivec_show/vshow** : display the content of the sample vector in PSUADE's workspace.

This command is to be used after an internal sample vector (register) has been created using ivec_create (or vcreate) to show the current content of the register.

## 2.5 Commands for RS-based Uncertainty/Sensitivity Analysis

This section describes commands for using the 'validated' response surface in computing various quantitative uncertainty and sensitivity measures.

**rsua** : perform uncertainty analysis using a response surface.

This command performs uncertainty analysis on the response surface (constructed from the loaded sample). As such, it requires another evaluation sample (so this command assumes that all sample inputs are uncertain) for propagating uncertainty through the response surface (the evaluation sample should be large enough to enable the accurate calculation of summary statistics). Users have the option to provide their own evaluation sample or request PSUADE to create one. If a stochastic response surface is selected (e.g. GP, Kriging, MARSB, regression), response surface uncertainties will also be shown in the PDF and CDF plots. The result of this analysis will be stored in a file called 'rsua_sample'. A Matlab (or Scilab) file called 'matlabrsua.m' (or scilabrsua.sci) will have been created for visualization at the completion of this command.

**rsuab** : perform uncertainty analysis using bootstrapped response surfaces.

This command is similar to rsua. The main difference is that this command estimates response surface uncertainties by first drawing many (say $n = 50$) bootstrapped samples from the loaded sample, constructing $n$ response surfaces, and computing $n$ statistics one from each bootstrapped response surfaces. An evaluation sample (e.g. those generated by *gensample*) in PSUADE data format will be requested from users to be used to propagate uncertainties through the response surfaces. A Matlab (or Scilab) file called 'matlabrsuab.m' ('scilabrsuab.sci') will have been created for visualizing the PDFs and CDFs at the completion of this command.

**rsmeb** : perform main effect sensitivity analysis using bootstrapped response surfaces.

This command performs first-order sensitivity analysis (Sobol' indices or main effects) on the bootstrapped response surfaces (similar to rsuab). This command internally creates a sample for use in computing the main effects. Users can provide their desired input distributions (in the 'INPUT' section) and turn on the 'use_input_pdfs' in the 'ANALYSIS' section so that PSUADE knows to use the user-specified distribution for analysis. This method performs the same analysis as rssobol1b but uses a different algorithm. At the end of this analysis, a Matlab file called 'matlabrsmeb.m' (or a Scilab file called 'scilabrsmeb.sci') will have been created.

**rsieb** : perform second-order sensitivity analysis using bootstrapped response surfaces.

This command is similar to rsmeb except that it performs second-order sensitivity analysis (Sobol' indices for two-way interaction effects) on the bootstrapped response surfaces. This method performs the same analysis as rssobol2b but uses a different algorithm. At the end of this analysis, a Matlab file called 'matlabrsieb.m' (or a Scilab file called 'scilabrsieb.sci') will have been created.

**rssobol1** : perform variance-based main effect analysis using a response surface.

This command computes first-order Sobol' indices for the response surface built from the loaded samples. (Make sure to validate your response surfaces before performing this analysis.) It performs the same function as rsmeb except that it uses a different algorithm and no bootstrapping is used for RS construction. Also, constraints can be imposed by using 'rs_constraint' in the ANALYSIS section of the sample file. At the end of this analysis, a Matlab file called 'matlabrssobol1.m' (or a Scilab file called 'scilabrssobol1.sci') will have been created.

**rssobol2** : perform variance-based second-order analysis using a response surface.

This command computes second-order Sobol' indices for the response surface built from the loaded samples. It performs the same function as rsieb except that it uses a different algorithm and no bootstrapping is used. Also, constraints can be imposed by using 'rs_constraint' in the ANALYSIS section of the PSUADE sample file. At the end of this analysis, a Matlab file called 'matlabrssobol2.m' (or a Scilab file called 'scilabrssobol2.sci') will have been created.

**rssoboltsi** : perform variance-based total-order sensitivity analysis using a response surface.

This command computes total-order Sobol' indices for the response surface built from the loaded samples. Constraints can be imposed by using 'rs_constraint' in the ANALYSIS section of the PSUADE sample file. At the end of this analysis, a Matlab file called 'matlabrssoboltsi.m' (or a Scilab file called 'scilabrssoboltsi.sci') will have been created.

**rssobolg** : perform variance-based group sensitivity analysis using a response surface.

This command computes group-order Sobol' indices for the response surface built from the loaded samples. Users need to provide a file to specify the groupings (that is, how the sample inputs are grouped), the format of which will be shown when this command is run. Constraints can be imposed by using 'rs_constraint' in the ANALYSIS section in the sample file. No visualization file will be created.

**rssobol1b** : perform variance-based main effect analysis using bootstrapped response surfaces.

This command is the same as rssobol1 except that it uses bootstrapped response surfaces to estimate response surface uncertainties when computing first-order Sobol' indices. At the end of this analysis, a Matlab file called 'matlabrssobol1b.m' (or a Scilab file called 'scilabrssobol1b.sci') will have been created.

**rssobol2b** : perform variance-based second-order effect analysis using bootstrapped response surfaces.

This command is the same as rssobol2 except that it uses bootstrapped response surfaces to estimate response surface uncertainties when computing second-order Sobol'

indices. At the end of this analysis, a Matlab file called 'matlabrssobol2b.m' (or a Scilab file called 'scilabrssobol2b.sci') will have been created.

**rssoboltsib** : perform variance-based toal-order effect analysis using bootstrapped response surfaces.

This command is the same as rssoboltsi except that it uses bootstrapped response surfaces to estimate RS uncertainties when computing total-order Sobol' indices. At the end of this analysis, a Matlab file called 'matlabrssoboltsib.m' (or a Scilab file called 'scilabrssoboltsib.sci') will have been created.

**aeua** : perform response surface-based mixed aleatory-epistemic uncertainty analysis.

This command performs outer-inner iterations to create cumulative distribution functions for mixed aleatoric-epistemic uncertainties. A Matlab file will be created ('matlabaeua.m' or 'scilabaeua.sci') at the end for visualing the 'p-box'.

**so_ua/ua** : perform response surface-based second-order uncertainty analysis.

This command performs second-order uncertainty analysis quantifying the effect of uncertainties in the distributions of the input parameters. Specifically, if an input variable has a uniform distribution but the lower and upper bounds are uncertain (for normal distribution, the mean and/or the standard deviation may be uncertain), this command evaluates output variability as the result of this input distribution uncertainty. At the end, A Matlab file called 'matlabsoua.m' (or a Scilab file called 'scilabsoua.sci') will have been created.

## 2.6 Commands for Optimization/calibration

This section describes commands for using parameter estimation using statistical inferences as well as multi-objective optimization. Other optimization methods are available via the batch mode.

**rsmcmc** : perform inference using a response surface.

This command should be used after a PSUADE sample has been loaded into PSUADE's workspace. It performs a parameter inference using the response surface built from the loaded sample (make sure to validate the response surface first). Users will have to answer a few questions (including options to add a discrepancy model and to generate a posterior sample) to set up the inference problem. Upon completion, the file 'matlabmcmc2.m' (or 'scilabmcmc2.sci') will have the posterior plots and 'MCMCPostSample' will have the posterior samplei for use in, e.g., mcmc_predict.

**mcmc_dm** : discrepancy modeling for MCMC.

The rsmcmc command may be used with discrepancy modeling. However, to avoid confounding the discrepancy model with the likely parameter values, rsmcmc requires users to provide the 'likely' parameter values. Sometimes these 'likely' values may be

unknown. Usually users simply use the modes of the parameters or the midpoints within the parameter ranges as 'likely' values. This command provides another way of finding these 'likely' values - by searching the calibration parameter space to find a 'nominal' point that minimizes certain metric.

**mcmc_predict** : predict new sample points by propagating MCMC posterior samples throught a response surface constructed from the loaded sample.

This command is another name for rsevaluate2 (so details are given under that command).

**mcmc** : perform inference using a simulator.

This command should be used after a PSUADE input file (and no need to load a sample file because it relies on the simulator, just the definition of inputs and outputs) has been loaded into PSUADE's workspace. It performs statistical inference on the simulator pointed to by the 'driver' (in the 'APPLICATION' section) in the loaded PSUADE input file. Since inference requires hundreds to thousands of simulations, it is time-consuming even when the simulators are fast to run (due to I/O operations, as PSUADE uses file-based protocol with the simulators). Users will have to answer a few questions to set up the inference problem. Upon completion, the file 'matlabmcmc2.m' (or 'scilabmcmc2.sci') will have the posterior plots and 'MCMCPostSample' will have the posterior samplei for use in mcmc_predict.

**mo_opt** : response surface-based multi-objective optimization.

This command is useful if you have more than one sample output and the objective function (for optimization) is formed by weighted combination of these outputs (rather than the Pareto front) where the weights are unknown. So given the ranges of these weights, this command generates optimal solution at the lattice points in the weight space via running many optimization at different combinations of the weights.

**set_rsmcmc_method** : select which inference method to use.

There are a few options for which inference algorithm to use in rsmcmc and this command allows users to choose which method to use.

## 2.7 Commands for Creating Visualization Plots

This section describes commands for visualizing the loaded raw sample as well as response surfaces constructed from the loaded raw sample.

**splot** : generate scatter plots of an output against each input.

This command creates multiple Matlab/Scilab scatter plots (in 'matlabsp.m' or 'scilabsp.m') with respect to each input on the X-axis and a user-selected output on the Y-axis. This command is useful for visualizing how the sample outputs vary with respect to each individual input.

**splot2** : generate scatter plots of an output against 2 inputs.

This command creates a Matlab/Scilab scatter plot with 2 selected inputs on the X- and Y-axes, and an output on the Z-axis (in 'matlabsp2.m' or 'scilabsp2.m'). This command is useful for investigating problems with fitting a 2-parameter model with response surfaces via visualizing the actual simulation output behavior. It is not very useful when the simulation model under consideration has more than 2 inputs (for 3 inputs, use splot3).

**splot3** : generate scatter plots for 3 inputs.

This command creates a Matlab scatter plot ('matlabsp3.m') with 3 selected inputs on the X-, Y-, and Z-axes; and a selected output represented by the sizes of dots. This command is useful for investigating problems with fitting a 3-parameter model with response surfaces via visualizing the actual simulation output behavior. It is not useful when the simulation model under consideration has more than 3 inputs.

**splot3m** : generate scatter plots for 3 inputs (movie mode).

This command creates a sequence of Matlab scatter plots (in 'matlabsp3m.m') with 2 selected inputs on the X-, Y-axes, a third input on the time axes, and a selected output on the Z-axis. This command works only for 3-input factorial designs. It is useful for visually inspecting the loaded sample. It is not useful when the loaded sample has more than 3 inputs, or when the sample points are not on a lattice (but you can create a lattice sample by evaluating a response surface with a factorial sample before using this command).

**rs1** : create an one-input-one-output response surface plot.

This command creates a X-Y response surface plot with X, Y being an user-selected input and output. All other input variables can be set to some default values (mid values of their ranges) or set by users. Matlab or Scilab plots ('matlabrs1.m' or 'scilabrs1.sci') are available.

**rs1s** : create an one-input-one-output response surface plot with uncertainty.

This command creates a X-Y response surface plot with X, Y being an user-selected input and output, and interpolation uncertainties are also shown (thus, a fuzzy response surface such as GP is appropriate). All other input variables can be set to some default values (mid values of their ranges) or set by users. Matlab or Scilab plots are available.

**rs2** : create an two-input-one-output response surface plot.

This command creates a X-Y-Z response surface plot with the X and Y axes being 2 user-selected inputs, and the Z-axis an user-selected output. All other input variables can be set to some default values (mid values of their ranges) or set by users. Matlab or Scilab plots are available.

**rs3** : create an three-input-one-output response surface plot.

This command creates a X-Y-Z response surface plot with the X, Y, and Z axes being 3 user-selected inputs, and the magnitudes of the selected output values are represented by a color chart. The plot file is currently only available in Matlab format ('matlabrs3.m').

**rs3m** : create an three-input-one-output response surface plot (in movie mode).

This command creates a X-Y-Z-T response surface plot with the X and Y axes being 2 of the 3 user-selected inputs, and the Z-axis an user-selected output. The third input is represented by the time axis T. The plot file is currently only available in Matlab format ('matlabrs3m.m').

**rs4** : create an four-input-one-output response surface plot (movie).

This command creates a X-Y-Z-T response surface plot with the X, Y, and Z axes being 3 of the 4 user-selected inputs, and the time axis T the fourth input. The magnitudes of the selected output is represented by a color chart. The plot file is currently only available in Matlab format ('matlabrs4.m').

**rssd** : create a response surface error plot.

This command creates a 1 to 4-input (depending on how many inputs are in the sample) response surface error (the errors are prediction uncertainties) plot. The response surface methods used here for estimating the interpolation error must be one of Gaussian process, MARS with bootstrapped aggregation, Kriging, or polynomial regression (other response surfaces do not provide error estimates). This command will create a Matlab file called 'matlabrssd.m' showing response surface uncertainties in the input space.

**rssd_ua** : perform uncertainty analysis of response surface errors.

This command first uses the loaded sample to create a response surface (Gaussian process, MARS with bagging, Kriging, or polynomial regression). It then estimates the prediction errors of the response surface over the input space by using a large space-filling sample. Basic statistics such as mean and standard deviation will be computed. This command is useful to analyze distributions of prediction errors caused by the selected response surface. This command creates a Matlab file called 'matlabua.m' showing the distribution of response surface uncertainties.

**rsi2** : create an two-input-multiple-output response surface intersection plot.

This command uses two sample inputs and two or more user-selected outputs (say, m) to build m response surfaces, then applies constraints (1 per output) to the response surfaces, and finally creates a Matlab plot showing the feasible region in the two-input space that satisfies the constraints. A Matlab file ('matlabrsi2.m') will be created for visualization.

**rsi3** : create an three-input-multiple-output response surface intersection plot.

This command is similar to rsi2 except that it creates a three-input-m-output intersection plot. The intersection plot can be visualized in Matlab ('matlabrsi3.m').

**rsi3m** : create an three-input-multiple-output response surface intersection plot in movie mode.

This command is similar to rsi3 except that the third input is represented by the time axis. The intersection plot can be visualized in Matlab ('matlabrsi3m.m').

**rawi2** : create an two-input-multiple-output intersection plot using raw sample data.

This command is similar to rsi2 except that it assumes the sample is already a factorial design and thus no response surface interpolation is required. This plot is currently available in Matlab format only ('matlabrawi2.m').

**rawi3** : create an three-input-multiple-output intersection plot using raw sample data.

This command is similar to rawi2 except that it creates a three-input intersection plot. This plot is currently available in Matlab format only ('matlabrawi3.m').

**rspairs** : create response surfaces for all 2-input pairs.

This command creates a number of sub-plots each of which is a 2-input response surface plot (by freezing all other inputs at their nominal values) for all pairs of the selected inputs. The plots are currently available in Matlab format only ('matlabrspairs.m').

**rsipairs** : create intersection plots for all 2-input pairs.

This command creates a number of sub-plots each of which is a 2-input response surface plot for all pairs of the selected inputs while holding all other inputs fixed. Constraints will be applied to all outputs. The plots are currently available in Matlab format only ('matlabrsipairs.m').

**iplot1** : create a one-input scatter plot.

This command creates a sample input scatter plot for a selected input (sample number on the X-axis and input values on the Y-axis). It creates a Matlab file called 'matlabiplt1.m' or Scilab file called 'scilabiplt1.sci'.

**iplot2** : create a two-input scatter plot.

This command creates a sample input scatter plot for two selected inputs (input 1 on the X-axis and input 2 on the Y-axis). It creates a Matlab file called 'matlabiplt2.m' or Scilab file called 'scilabiplt2.sci'.

**iplot3** : create a three-input input scatter plot.

This command creates a sample input scatter plot for three selected inputs (on X-, Y-, and Z-axes). It creates a Matlab file called 'matlabiplt3.m' or Scilab file called 'scilabiplt3.sci'.

**iplot4m** : create a four-input response surface plot (in movie mode).

This command creates a sample input scatter plot for four selected inputs (on X-, Y-, Z-, and T-axes), that is, the fourth input is represented by the time axis. It creates a Matlab file called 'matlabiplt4m.m'.

**iplot2_all** : create two-input scatter plots for all input pairs.

This command is similar to iplot2 except that the same operation is performed on all input pairs. A Matlab file called 'matlabiplt2_all.m' or Scilab file called 'scilabiplt2_all.sci' will be created for visualization.

**iplot_pdf** : plot sample PDFs for individual inputs.

This command creates histogram plots for each selected input of the loaded sample. A Matlab file called 'matlabiplt_pdf.m' or Scilab file called 'scilabiplt_pdf.sci' will be created for visualization.

**iplot2_pdf** : plot sample PDF for all inputs and all input pairs.

This command creates histogram plots for each input of the loaded sample and heat maps for all input pairs. A Matlab file called 'matlabiplt2_pdf.m' or Scilab file called 'scilabiplt2_pdf.sci' will be created created for visualization.

**oplot2** : create a two-output scatter plot.

This command creates a scatter plot for two selected outputs (output 1 on the X-axis and output 2 on the Y-axis). A Matlab file called 'matlaboplt2.m' or Scilab file called 'scilaboplt2.sci' will be created for visualization.

**oplot_pdf** : plot sample PDFs for all outputs.

This command creates histogram plots for each output of the loaded sample. At the end, a Matlab file called 'matlaboplt_pdf.m' or a Scilab file called 'scilaboplt_pdf.sci' will be created for visualization.

**oplot2_pdf** : plot sample PDFs for all outputs and all pairs of outputs.

This command creates histogram plots for each output of the loaded sample and heat maps for all output pairs. A Matlab file called 'matlabiplt2_pdf.m' or a Scilab file called 'scilaboplt2_pdf.sci' will be created for visualization.

**ihist** : create an input histogram.

This command takes any single input and creates a histogram that can be displayed by Matlab ('matlabihist.m' or 'scilabihist.sci'). This command is useful to examine distribution of individual inputs from a posterior sample (e.g. MCMCPostSample loaded using iread).

**ihist2** : create an 2-input histogram.

This command takes any two inputs and creates a 3D histogram that can be displayed by Matlab ('matlabihist2.m' or 'scilabihist2.sci'). This command is useful to examine distribution of any two inputs from a posterior sample (e.g. MCMCPostSample loaded using iread).

**iotrace** : plot inputs/output trajectories of all sample points.

This command produces a plot with the number of lines (colored) equal to the sample size. The Y-axis goes from 1 to $n + m$ where $n$ and $m$ are the numbers of selected inputs and outputs, respectively. The X-axis corresponds to the actual input and output values. Each line goes from $Y = 1$ to $Y = n + m$ where the X value at $Y = i$ being the value of the i-th input for the sample point or the value of the $i - n$ output if $i > n$. The plot may give insights of patterns of input-output behaviors.

## 2.8 Commands for Setting Up PSUADE Runs

This section describes commands for setting up the scripts and input files before launching PSUADE runs.

**setupguide** : display a short guide to help set up an application.

This command gives a step-by-step instructions on how to set up PSUADE to perform uncertainty quantification on your models.

**geninputfile** : create a PSUADE input file.

PSUADE requires an input file to generate samples and run simulations (and optionally performing analysis). This command guides users in creating this input file (which has input information such as input dimension, input ranges, input distribution type, and input name; output information; sampling information such as sampling design and sample size; simulation information; and analysis information.

**genbatchfile** : create a batch file.

Since many applications at LLNL use our Livermore Computing machines where jobs are submitted in batch mode, this command helps to set up the batch jobs. A batch script will be created that users can modify the run information such as number of processors and time limit for the LLNL 'msub' job scheduler.

**gendriver** : create a PSUADE driver program.

A driver program is needed in a PSUADE input file. This driver program takes the input parameter values from a file (argument 1), substitutes the data into the application input decks, runs the application, post-processes the application outputs, and writes the output data to another specified file (argument 2). This command helps to create this driver program.

**genexample** : create a PSUADE example.

This command creates a simple PSUADE example. After the example has been created, simply run psuade psuade.in to see the results.

**chkjobs** : monitor the status of job runs.

This command helps users check the status of their simulation runs. Just follow instructions in this command for checking (you have to specify what the indicators are for job completion or failures).

## 2.9 Commands for Editing the Sample in PSUADE's Workspace

This section describes commands for manipulating the resident sample in PSUADE's workspace.

**svalidate** : change the states of selected sample points to be 'evaluated'.

Every sample point loaded to PSUADE's workspace has a status flag. When the sample point has been evaluated (by running a simulation), this status flag will be set to '1'. This command forces this flag to be set to '1' for the selected sample points. Once the flag is set, an 'evaluated' sample point will not be evaluated when PSUADE is run on this sample.

**sinvalidate** : change the state of selected sample points to be 'unevaluated'.

This command does the opposite of svalidate, namely, to switch the status flag of the selected sample points to '0'. This command is useful when some sample points are to be re-run (sample points with status flag= 1 will not be evaluated).

**srandomize** : randomly change the order of the sample points.

This command randomly permutes the sample points already loaded in PSUADE's workspace.

**iadd1** : create a new sample input in PSUADE's workspace.

This command creates a new sample input to be appended (to be the last input) to the current input set in PSUADE's workspace. The values of this new input in the sample are drawn randomly from $[0, 1]$. This command is useful in adding a nugget input variable to the existing sample. This new input can be modified using other edit commands.

**oadd1** : create a new sample output in PSUADE's workspace.

This command creates a new sample output to be appended to the sample outputs in PSUADE's workspace. The values of this new output in the sample are drawn randomly from $[0, 1]$. This command is useful in association with the oop command (e.g. add a new output, set the output to be the sum of 2 other outputs).

**imodify** : modify an input of a selected sample point.

This command modifies a selected input of a selected sample point in PSUADE's workspace.

**omodify** : modify an output of a selected sample point.

This command modifies a selected output of a selected sample point in PSUADE's workspace.

**ifilter** : remove sample points that do not satisfy certain input conditions (bounds).

This command deletes all sample points with the values of a selected input falling outside some user-specified lower and upper bounds.

**ofilter** : remove sample points that do not satisfy certain output conditions.

This command deletes all sample points with the values of a selected output falling outside some user-specified lower and upper bounds.

**sfilter** : remove sample points that have already existed in another user-specified sample file.

This command asks users for another sample file (in PSUADE data format) and then compares the loaded sample with this sample. If it finds matches, it deletes the 'matched' points from the loaded sample.

**idelete** : remove selected inputs from the sample.

This command removes a subset of inputs from the sample in PSUADE's workspace (that is, removing 2 inputs from a sample with 10 inputs will yield a 8-input sample).

**odelete** : remove an output from the sample.

This command removes a selected inputs from the sample in PSUADE's workspace (that is, removing one output from a sample with 10 output will yield a 9-output sample).

**sdelete** : remove a subset of points from the sample.

This command removes a selected subset of sample points from PSUADE's workspace. This command is is useful when some sample points have been determined to be outliers and you want to remove them from further analysis.

**ikeep** : keep selected inputs in the sample.

This command, as opposed to *idelete*, keeps a subset of inputs in the sample in PSUADE's workspace (that is, it removes inputs that are not selected.)

**okeep** : keep selected outputs in the sample.

This command, as opposed to *odelete*, keeps a subset of outputs in the sample in PSUADE's workspace (that is, it removes outputs that are not selected.)

**skeep** : keep selected sample points.

This command, as opposed to *sdelete*, keeps a subset of sample points in the sample in PSUADE's workspace (that is, it removes sample points that are not selected.)

**spurge** : remove failed sample points.

This command removes all sample points that have the values of at least one output being 'UNDEFINED' ($1.0e35$) or the status flag not equal to 1.

**ishuffle** : re-arrange the orders of sample inputs.

This command re-arranges the sample inputs into a different ordering specified by users. This command is useful in a multi-stage UQ analysis when the posterior sample file has a different input ordering than the one in the full system input list.

**oshuffle** : re-arrange the orders of sample outputs.

This command re-arranges the sample outputs into a different ordering specified by users.

**sshow** : display one sample point.

This command shows the sample input and output values for the selected sample point.

**sinfo** : display information about the current sample.

This command displays information about the loaded sample such as sample size, number of inputs, number of outputs, input names, and output names.

**list1** : list one output with one input of the loaded sample.

This command lists the values of one input and one output of all points in the loaded sample. Options are provided for sorting the inputs or the outputs before displaying. This command does not re-order the sample in PSUADE's workspace (for re-ordering, use isort or osort).

**list2** : list one output with two inputs of the loaded sample.

This command is similar to list1 except it displays two selected inputs instead of one. Options are provided for sorting the inputs or the outputs before displaying.

**listall** : list one output with all sample inputs.

This command is similar to list1 except it displays all inputs. An option is provided for sorting the outputs before displaying.

**isort** : sort the loaded sample based on the input values.

This command re-orders the sample points based on the values of the inputs. The first input is sorted (in ascending order) first, followed by the second input and so on. Once this command has been executed, PSUADE's workspace will have this new ordering.

**osort** : sort the loaded sample based on the values of a selected output.

This command re-orders (in ascending order) the sample points based on the values of a user-specified output. Once this command has been executed, PSUADE's workspace will have this new ordering.

**imax** : display the sample point with the maximum value of a selected input.

This command finds and displays the sample point in the loaded sample with the maximum value of a selected input.

**omax** : display the sample point with the maximum value of a selected output.

This command finds and displays the sample point in the loaded sample with the maximum value of a selected output.

**imin** : display the sample point with the minimum value of a selected input.

This command finds and displays the sample point in the loaded sample with the minimum value of a selected input.

**omin** : display the sample point with the minimum value of a selected output.

This command finds and displays the sample point in the loaded sample with the minimum value of a selected output.

**onorm** : display the 2-norm of a sample output.

This command computes and displays the 2-norm of a selected sample output (norm computed from all sample points in PSUADE's workspace).

**osum** : display the sample sum of a sample output.

This command computes and displays the sum of a selected sample output (sum from all sample points in PSUADE's workspace).

**inormalize** : normalize a selected sample input.

This command is for normalizing a selected sample input either to a desired range or to the standard Gaussian distribution.

**onormalize** : normalize a selected sample output.

This command is for normalizing a selected sample output either to a desired range or to the standard Gaussian distribution.

**iscale** : re-scale a selected input into a different range.

This command, along with 'iadd1' and 'ireset', is useful for creating a new categorical variable (e.g. Suppose you want to create a sample input randomly populated with the integers 0, 1, or 2. You can first use 'iadd1' to create a new input drawn randomly from $[0, 1]$, re-scale it to $[0, 3]$, and then to use 'ireset' to re-populate the sample input with

0, 1, or 2.) This command is also useful for local sample refinement by creating another sample with the same coordinates for all inputs except one selected input, which is to be shrunk to a smaller range (or expanded to a larger range). The 're-scaled' sample will be in PSUADE's workspace ready to be written to a file.

**oscale** : re-scale a selected output into a different range.

This command is not designed for a specific purpose. It is implemented to provide general flexibility of manipulating the loaded sample.

**ireset** : map a selected input with a given range of values to a distinct value.

This command is intended for converting continuous input variables into discrete (from a range to a single value).

**oreset** : map a selected output with a given range of values to a distinct value.

This command is intended for converting continuous output variables into discrete (from a range to a single value).

**ifloor** : map a selected input of the loaded sample to the nearest smaller integer.

This command is intended for converting continuous input variables into discrete values by setting the values to the nearest smaller integer.

**iceil** : map a selected input of the loaded sample to the nearest larger integer.

This command is intended for converting continuous input variables into discrete values by setting the values to the nearest larger integers.

**iround** : map a selected input of the loaded sample to the nearest integer.

This command is intended for converting continuous input variables into discrete values by setting the values to the nearest integers.

**itag** : tag and display sample points with a selected input falling outside some bounds.

Sample points in PSUADE's workspace with values of a selected input that falls outside the specified bounds will be tagged, and the untagged sample indices will be displayed at the end of this command. Calling this command again or otag will accumulate the tagged sample points. This command is useful for displaying a list of sample points with certain input properties (e.g. to display sample points with values of input 2 outside the range of $[0, 1], [5.6], and [9, 10]$, this command should be called 3 times). The tags are reset (the list will be re-initialized) after a load.

**otag** : tag and display sample points with a selected output failling outside some bounds.

This command is similar to itag except that it tags based on some selected output. This command works in conjunction with itag in accumulating the tagged sample points. The tags are reset (the list will be re-initialized) after a load.

**rm_dup** : take out duplicate sample points.

This command is useful to compress or remove duplicate sample points (points with same input values). It offers a few options for how to treat duplicate points. For example, duplicate sample points can be compressed by computing the mean, median, or standard deviation. Alternatively, duplicate points can be put into quantiles after applying kernel density estimation.

**sopt_mmd** : optimize inter-sample distances.

This command searches for the permutation of the resident sample inputs that gives the best space-filling property (maximize the minimum distance between any two sample points). This command uses the coordinate exchange algorithm, which preserves, for example, the Latin hypercube property (so one can first create a Latin hypercube sample and then run this command to optimize its space-fillingness).

**iop** : perform various operations on selected inputs.

This command performs different operations on selected inputs of all sample points in PSUADE's workspace, operations such as reciprocal, linear combination, exponentiation, logarithmic or trigonometric transformations.

**oop** : perform various operations on selected outputs.

This command performs different operations on selected outputs of all sample points in PSUADE's workspace, operations such as linear combination, exponentiation, or logarithmic.

**ioop** : modifies selected output to be some linear combination of the corresponding selected inputs.

This command forms a linear combination of two selected inputs and replaces a selected output with the results.

**setdriver** : set the driver fields of the current PSUADE sample.

Each sample file loaded to PSUADE's workspace has information other than the sample input and output values (e.g. sampling method, response surface type, drivers). This command is for modifying the 'driver' fields. The driver fileds are: simulation driver ('driver'), optimization driver ('opt_driver'), auxiliary optimization driver ('aux_opt_driver'), ensemble simulation driver ('ensemble_driver') and ensemble optimization driver ('ensemble_opt_driver'), the last of which is especially created to facilitate optimization under uncertainty (Refer to the 'APPLICATION' section of any 'psuadeData' file). Once this command has been executed, a 'write' command will have this updated driver information in the new file. Alternatively, this modified setup can be run using the run command to evaluate the sample using the user-specified driver.

## 2.10 Commands for Optimal Experimental Designs

This section describes commands for creating various types of optimal experimental designs. There are two classes of optimal experimental designs in this context. The first class assumes that all model inputs are design parameters (that is, no uncertain parameters), and prediction uncertainties, if needed for deriving optimal designs, are induced primarily by response surface prediction uncertainties. Falling into this first class are space-filling methods such as max-min distance and stochastic response surface-based method as minimax prediction variance. The second class, on the other hand, assumes some of the model inputs are design parameters and some are uncertain parameters, and prediction uncertainties that are used to compute performance metrics are induced by the probability distribution of the uncertain parameters. Falling into this second class are Bayesian optimal design and designs derived from properties of the Fisher information matrix.

**odoe_mmd** : select a subset of the loaded sample using the max-min distance criterion.

This command examines the sample (the candidate set) in PSUADE's workspace and selects a subset (batch size to be decided by users) with the property that the minimum distance between any two points is maximized among all possible combinations. This function is computationally prohibitive for large sample size and/or large batch size (the cost is $C(N, l)$ where $C$ is the combinatorial operator, $N$ is the sample size and $l$ is the batch size. Less expensive version is provided using global optimizer but it may give sub-optimal results. Optimal results can be obtained more efficiently using the hybrid method whereby an initial global optimization is performed to obtain some sub-optimal results, which may improve efficiency dramatically in the second exhaustive search stage. In addition, a weighted option is provided to allow users to indicate which sample points have larger weights (e.g. points with larger prediction uncertainties may be given larger weights) via some selected sample outputs. NOTE: this command assumes all inputs are used to compute the distance measure (that is, all inputs are design inputs and no input is an uncertain parameter), and the inputs have been scaled properly for distance calculation.

This command can actually be used in three ways:

1. Input-only space-filling design based on max-min distance approach using either global optimization or brute force search.
2. For input-output space-filling MMD design, move the selected outputs to the input section first (use a combination of odelete and modification to the PSUADE sample data file) and then run this command.
3. For weighted space-filling design, populate a selected output with prediction variances (by evaluating this sample with some response surface constructed from another sample), and use the weighted version of this command (Using prediction variances as weights amounts to the W-optimal design when one design is to be selected. With batch design, this is equivalent to a mixed space-filling and W-optimal design).

**odoe_mmv** : generate min-max design using Gaussian process to estimate prediction variances.

This method first creates a Gaussian process (GP) response surface from the loaded sample in PSUADE's workspace or, alternatively, from the GP hyperparameters specified by users. It then uses the GP covariance matrix in search of the optimal design. The design criterion is G-optimality, that is, choose the design that will minimize the maximum variances (from GP) in the evaluation sample (to be provided by users). For batch design, finding the G-optimal design is computationally prohibitive (combinatorial in the batch size). As such, this method offers a few options to reduce the overall cost in return for sub-optimal designs. NOTE: this command assumes all inputs are design inputs (that is, no input is an uncertain parameter).

**odoe_mav** : generate minimum average design using Gaussian process to estimate prediction variances.

This method is similar to sf odoe_mmv except that it uses the I-optimality (minimizing the average variance in the evaluation sample).

**odoe_dmetric** : compute D-metric from the loaded sample.

This command first computes an input covariance matrix from the sample inputs (that is, the sample outputs are ignored) in PSUADE's workspace. It then computes the D-metric by forming the product of all eigenvalues of the covariance matrix.

**odoe_ametric** : compute A-metric from the loaded sample.

This command is similar to odoe_dmetric except that it computes the A-metric by forming the sum of all eigenvalues of the covariance matrix.

**odoe_emetric** : compute E-metric from the loaded sample.

This command is similar to odoe_dmetric except that it computes the E-metric by finding the maximum eigenvalue of the covariance matrix.

**odoe_entropy** : compute entropy from the loaded sample.

This command computes the entropy of the probability distribution for the selected inputs represented by the loaded sample. It first forms histograms from the loaded sample and then computes the entropy.

**odoe_pv** : compute prediction variance.

This command computes for each point of a candidate set its prediction variance (prediction uncertainty is assumed to be from the response surface only so no uncertain inputs are assumed) using a response surface constructed from the loaded sample.

**odoeu_boptn** : finds candidate subset that optimizes certain optimality criterion.

This command (note the use of 'odoeu' instead of 'odoe') assumes that some of the sample inputs are design and some are uncertain parameters. It finds for a given design

candidate set the subset of size $n$ (specified by users) that optimizes either one of the G/I/D/A/E-measures. A prior sample is to be provided by users to represent the probability distribution of the uncertain inputs. The different measures are:

**G-measure** : the maximum post-inference prediction variance among all points in the evaluation sample when the n-tuple is used as a design.

**I-measure** : the average post-inference prediction variance among all points in the evaluation sample when the n-tuple is used as a design.

**D-measure** : the post-inference product of eigenvalues of the covariance matrix constructed from the posterior sample of the uncertain inputs when the n-tuple is used as a design.

**A-measure** : the post-inference sum of eigenvalues of the covariance matrix constructed from the posterior sample of the uncertain inputs when the n-tuple is used as a design.

**E-measure** : the post-inference maximum eigenvalue of the covariance matrix constructed from the posterior sample of the uncertain inputs when the n-tuple is used as a design.

This method uses global optimizer so the result may be sub-optimal. This method is slower than odoeu_foptn in general, but this method may be more suitable if users have better information on the experimental uncertainty (rather than deriving from the prior distribution of the uncertain parameters provided by users).

**odoeu_foptn** : finds candidate subset that optimizes certain Fisher-based optimality criterion.

This command is similar to odoeu_boptn except that it uses the Fisher information matrix-based approach to compute the G/I/D/A/E metrics. It uses global optimizer so the result may be sub-optimal. This method is faster than odoeu_boptn in general.

**odoeu_beval** : evaluate the G/I/D/A/E measures for each member of the candidate set.

This command computes for every member of the candidate set its G/I/D/A/E measures based on the Bayesian inference results. It can be useful for comparing between pre- and post-inference measures (both will be displayed during the execution of this command) to track uncertainty reduction due to the use of the candidate member.

**odoeu_feval** : evaluate the Fisher-based G/I/D/A/E measure for each member of the candidate set.

This command is similar to odoeu_beval except that it computes the G/I/D/A/E measures based on the Fisher information matrix approach.

**odoeu_bevaln** : evaluate the G/I/D/A measure for a given batch of designs.

This command computes the G/I/D/A/E measures for a given design set as a whole (that is, one G-value, one I-value, etc., for the entire design set.)

**odoeu_rsmcmc** : create a posterior sample using a given batch design.

This command performs inference based on using the user-provided design set as experiments (experimental output and uncertainty may be provided by users or computed from the training sample loaded before this command is executed) and a user-provided prior sample. It produces a posterior sample for the uncertain inputs (D-, A-, and E-measures can then be computed for this posterior sample). This command can be used in conjunction with odoeu_boptn iteratively to perform 'SEQUENTIAL' design of experiments (posteriors of this command are used as prior in odoeu_boptn, which produces another design that can be used by this command to compute the next posterior and so on).

**odoeu_rseval** : predict new sample points by propagating a sample representing probability distribution of uncertain parameters through a response surface.

This command is similar to the mcmc_predict command except that this command does not support discrepancy modeling.

## 2.11 Commands for High-dimensional Inference with KPCA

For models with high-dimensional inputs whereby the inputs are correlated, a different (than the parameter screening described above) dimension reduction is warranted. This section describes dimension reduction based on kernel principal component analysis (KPCA) for random input fields as well as inference on the reduced space.

**kpca_create** : create a KPCA model from snapshots.

This command takes an ensemble of, say, $M$ snapshots each with dimension $n$, the desired KPCA scheme (from first-order to fifth-order), and the desired reduced dimension $r \ll n$; and perform eigen-analysis to construct a KPCA model in $r$ dimension that can be used for forward and inverse transformation between the original physical space (of dimension $n$) and the reduced feature space (of dimension $r$) during inference (mcmc_with_dr). The model will be stored in a user-specified file for future use.

**kpca_forward** : transform physical inputs into reduced feature inputs.

This command must be performed after kpca_create has been called. The KPCA model file produced by kpca_create will be used to transform the physical inputs (with dimension $n$) in a user-specified file to the feature inputs (dimension $r$), which will be written to another file.

**kpca_inverse** : transform reduced feature inputs back into physical inputs.

This command must be performed after kpca_create has been called. The KPCA model file produced by kpca_create will be used to transform (inverse) the feature inputs (with dimension $r$) in a user-specified file back to the physical inputs (dimension $n$), which will be written to another file.

**kpca_server** : run kpca_inverse as a server to an external request.

This command can be used by an external program running statistical inference which requires KPCA transformation provided by PSUADE. So the the external program makes request to PSUADE, waits for results, and continues processing until the next request is needed.

**gen_likelihood** : create Python script to compute likelihood given a feature vector.

This command helps users create a Python script that can be used in mcmc_with_dr for computing likelihood using KPCA for dimension reduction.

**mcmc_with_dr** : Perform MCMC (Metropolis-Hastings) with KPCA-based dimension reduction.

This command calls the Metropolis-Hastings MCMC using likelihood function created by gen_likelihood.

**opca** : perform principal component analysis on the sample outputs.

This command performs principal component analysis the sample outputs (that is, normalize each output and then apply singular value decomposition). Specifically, given the output matrix $Y$, the following decomposition is performed:

$$\hat{Y} = USV^T$$

where $\hat{Y}$ is the (normalized) sample output matrix with $N$ rows (sample size) and $m$ columns (number of outputs), $U$ is a matrix of size $N \times m$ (left singular vectors), $S$ is a diagonal matrix of size $m \times m$, and $V$ is a matrix of size $m \times m$ (right singular vectors). Upon exit from this command, the file 'psPCA.out' will contain the projection of the sample outputs onto the principal components $V$ (that is, $\hat{Y} \times V$ or $U \times S$). If you decide to use these principal components, you can use oreplace to insert the file 'psPCA.out' to your PSUADE sample file.

## 2.12 Other Miscellaneous Commands

There are a lot other commands that may be useful for users. This section describes these commands.

**rename** : rename a file.

This command is similar to the Linux 'mv' command. It is provided here for convenience since the load command may complain about using psuadeData as a data file so one may want to change the name of this file before loading.

**run** : run a PSUADE input file.

This command (e.g. 'run psuade.in') is equivalent to running the batch mode 'psuade psuade.in' but it is now run in the command line interpreter.

**quit** : exit the current PSUADE session.

**getstatus** : get the status of the previous command.

This command can be used to display the status the previous command - whether the previous command was successful or not.

**rsmax** : change the maximum sample size for response surface.

Response surface construction in generally is computationally intensive. Thus, for large sample sizes, the processing time can be prohibitive. As such, PSUADE internally sets a maximum sample size for response surface construction so that users can be warned. The default is 10000. This command is provided for users to override this restriction.

**sys** : initiate Linux shell command.

This command calls the external shell command with a system call. The string after this command will be passed to the shell for execution. for example, running sys ls on Linux will list the content of the current working directory.

**printlevel** : change the diagnostics print level.

Depending on the diagnostics level, different types of information are displayed on the console. For example, raising the 'printlevel' (e.g. to 3 or 4) will provide more information during response surface cross validation.

**output_file** : set the default PSUADE data file name.

The default PSUADE data file name is 'psuadeData'. This command allows users to change this default name.

**sqc** : check sample quality.

This command checks the quality of the loaded sample by computing the max-min distances between all pairs of sample points and other metrics (For example, the minimum distance between any 2 sample points should be as large as possible).

**ssc** : check sample smoothness.

The sample smoothness is measured by the average gradient of all individual points with their neighbors (with the number of neighbors defined by users). This command provides a rough check of smoothness of the sample output with respect to the inputs.

**nna** : perform nearest neighbor analysis for outlier detection.

This command finds the nearest neighbor for each sample point and then computes the difference between its output and the output of its nearest neighbor. If the gradient of a sample point with respect to its nearest neighbor (for a selected output) is large, it may indicate an 'outlier'.

**setranseed** : set the random number generator seed to a user-specified value.

This command is useful for 'repeatability' of analysis as many statistical analyis methods in PSUADE require drawing random samples.

**showformat** : display different PSUADE file formats.

User-provided information are often needed in design and analysis. These information are provided by users to PSUADE at various stages in specific data formats. This command lists several such file formats.

**scilab** : toggle between Matlab or Scilab graphics outputs.

PSUADE generates many graphical results in either Matlab or Scilab format. This command should be called before analysis is performed. Currently, Matlab graphics are better supported than Scilab graphics in PSUADE. However, for most Matlab scripts, conversion to run with Scilab requires only small changes.

**start_matlab** : start Matlab without leaving the PSUADE command line interpreter.

This command is mainly for convenience (view graphics without leaving the PSUADE command mode).

**checkformat** : check a user file for correct format.

There are currently several different types of data files used in various analysis. This command helps users check whether a data file has valid format.

**set_sam_method** : set sampling method.

This command changes the 'sampling_method' variable in the loaded sample. Any 'write' operation performed after this command will store the new sampling method to the sample file (in PSUADE format).

## 2.13 Advanced Commands

The commands in this section are more for advanced users who would like to perform various tasks such as adaptive sample refinement, finetuning different analysis methods, and others.

**script <file>** : where <file> is a PSUADE command file.

This command is equivalent to running PSUADE with <file> as an argument (which allows you to execute a batch script without leaving the PSUADE interactive session).

**io_expert** : toggle the I/O expert mode.

This expert mode affects I/O operations internal to PSUADE via prompting users for more questions.

**rs_expert** : toggle the response surface expert mode.

This expert mode affects response surface methods internal to PSUADE via prompting users for more detailed information about parameter selections in response surface analysis.

**rs_codegen** : toggle the response surface code generation mode.

Many response surface methods in PSUADE produce stand-alone Python and/or C codes for interpolation (that is, codes that can be plugged into other user codes). For example, turning on this mode before rscheck using Gaussian process (GP) will produce a file with a C program for GP interpolation on new points.

**ana_expert** : toggle the analysis expert mode.

This expert mode affects analysis methods internal to PSUADE via prompting users for more detailed information (e.g. option to perform logarithmic transformations on the inputs and outputs).

**sam_expert** : toggle the sampling expert mode.

This expert mode affects the internal sample generation methods by prompting users for more detailed information (e.g. how much space-fillingness is desired for the Latin hypercube sample).

**opt_expert** : toggle the optimization expert mode.

This expert mode affects optimization methods internal to PSUADE via prompting users for more detailed information (e.g. to turn on mixed-integer optimization in the SCE optimizer).

**genhistogram** : create scenarios based on a sample

This command takes sample, which has been loaded to PSUADE's local memory, and converts it to a smaller sample set where each sample point is associated with a probability. This command is useful for optimization under uncertainty: reduce a large sample from, for example, a posterior sample from MCMC to a smaller set of scenarios.

**genhistogram2** : create scenarios based on a sample and target size

This command is similar to genhistogram except that users are prompted for the target number of distinct sample points.

**genconfigfile** : create a PSUADE configure file.

Configuration files are useful for setting more detailed options (as in expert modes) non-interactively. This command spits out an example configuration file. Features can be activated by uncommenting the lines, and these changes can be imported to PSUADE via the useconfigfile command (or via a PSUADE input file).

**useconfigfile** : tell PSUADE to use a configure file.

Configuration files are useful for setting more detailed options (as in expert modes) non-interactively. This command tells PSUADE to use the given configure file for setting more detailed internal options.

**showconfigtable** : display the content of the configure table.

Once PSUADE has been started, a configure table will also have been created. Users can use this command to find out what are already set up in the configure table.

**setconfigoption** : set options in the configure table.

Once PSUADE has been started, a configure table will also have been created. Users can use this command to add entries to the configure table so that various commands can look up this table for certain analysis options. This command is useful when users want to set a certain analysis option but don't bother having to enter that option every time the analysis is performed. options (as in expert modes) non-interactively.

**resetconfigtable** : clean out the configure table.

Once PSUADE has been started, a configure table will also have been created. Users can use this command to add entries to the configure table so that various commands can look up this table for certain analysis options. At some points users may want to disable all options set earlier. This command facilitates this pruning operation.

**urefine** : refine a sample uniformly.

This command activates the sampling refinement operation on the loaded sample based on the original sampling method used to create it. For example, if the loaded sample was created using Latin hypercube, then the refinement method for Latin hypercube will be used for refinement. After this command, The 'enlarged' sample, which should be about twice the size of the original sample, will be resident in PSUADE's workspace and it can be exported to another PSUADE file for further processing. This command operates only on a few sampling designs (LH, MC, LPTAU, METIS), e.g., it does not make sense to refine a Morris sample.

**arefine** : refine a sample adaptively.

This command activates the adaptive sampling refinement operation on the loaded sample by subdividing the subdomains that give the largest prediction uncertainties from the response surface (as such, stochastic response surface methods such as Kriging are needed). The enlarged sample can be written to another PSUADE file for further processing. This command operates on any initial sample, but prefers the initial sample to be somewhat space-filling. Cubic splines (MARS) with bootstrapped aggregation will be the default method used to estimate where refinement should be performed. Other methods are available when the response surface expert mode is turned on.

**arefine_metis** : refine a METIS sample adaptively.

This command is similar to arefine except that it works only with an initial METIS sample with the METIS information having been stored in the 'psuadeMetisInfo' file (arefine works with any sampling designs).

**arefine_cv** : refine a sample adaptively.

This command activates the adaptive sampling refinement operation on the loaded sample by subdividing the subdomains containing sample points that give the largest cross validation errors from the response surface (stochastic response surface methods is not needed). The enlarged sample can be written to another PSUADE file for further processing. This command operates on any initial sample, but prefers the initial sample to be somewhat space-filling.

**arel_fact** : perform adaptive refinement on factorial designs fo reliability.

This command performs adaptive refinement along the boundary in the input space where the response surface interpolation gives values close to a user-defined threshold. This command uses regular lattice for refinement and so may be limited to low dimensional inputs.

**arel_fact2** : similar to arel_fact but for two inputs only.

**arel_metis** : similar to arel_fact but for using METIS instead of lattices.

This command is similar to arel_fact except that, instead of using regular lattices, it uses METIS-generated subdomains so that higher dimensional inputs can be handled more efficiently.

**moat_adjust <file>** : where <file> is the file that has the Morris adjustment data.

The Morris design may require some inputs to be adjusted due to the presence of constraints that cause the sample space to be non-hyper-rectangular. To create a Morris sample that stays inside the feasible parameter space, the moatgen command should be used first. The outputs from moatgen are to be used to adjust a Morris sample. So the steps to create a Morris sample given some constraints are:

1. create a standard Morris sample (in the hyper-rectangular space)
2. use moatgen to create the adjust file (say, adjustData)
3. change the second line of the adjust file to match up the input numbers
4. start PSUADE command line mode and load the standard Morris sample
5. use this command (moat_adjust adjustData)
6. write the adjusted sample to a file (use write)

**gmoat_adjust <file>** : where <file> is the file that has the Morris adjustment data.

This command is similar to moat_adjust except that the original sample should be a generalized Morris (GMOAT) sample.

**moatgen** : generate a Morris adjustment design.

This command creates a Morris adjustment file. More detail is found in the description of moat_adjust above. A PSUADE data file needs to be loaded first before moatgen can be used. The loaded sample is used to provide the constraints for creating the adjustment file. This command will ask for the desired resolution, which determines the width of the base where gradients will be computed (default is 4).

**moatgen2** : generate a Morris adjustment design for multiple constraints.

This command is similar to moatgen except that it can handle multiple constraints. This command does not require a PSUADE data file be pre-loaded (but at least a PSUADE input file needs to be pre-loaded). Constraint files will be requested. The format for the constraint file can be found in the showformat command.

**moat_concat <file>** : concatenate two Morris samples.

This command concatenates two Morris samples with two different sets of inputs. The two samples should have the same number of replications. For example, if the sample in the workspace has 10 inputs and 110 sample points and <file> has 5 inputs and 60 sample points (number of replications = 10), then the combined sample will have $10 \times (10 + 5 + 1) = 160$ sample points. The combined sample will be in the workspace ready to be written, for example, using the write command.

# 3 PSUADE Function Calls

PSUADE functions can be called directly from a user program. There are a few requirements that must be met in order to do this:

- PSUADE has to be compiled in shared library mode (meaning that when 'ccmake' is launched, the 'BUILD_SHARED' flag should be on). After compilation is completed, a number of PSUADE library files ('*.so') will be present in the 'build/lib' directory.
- In the user program that intends to use PSUADE functions, make sure to include the relevant include files. During compilation, make sure to set the proper include directory (that is, the '-I' option in C/C++ compilers). It is preferable to copy all PSUADE include files in a single directory (e.g. go to '<PSUADE.dir>/build', issue 'cp ../Src/*/*.h lib' where <PSUADE.dir> is the PSUADE top level directory). so that you only need to specify a single include directory.

```
${CPP} -c -I/<psuade.dir>/build/include main.cpp -o main.o
```

- When linking your user program to the PSUADE libraries, do (e.g. on Linux):

```
${CPP} -o main main.o <psuade.lib>/libpsuade.so <psuade.lib/libcobyla.so \
    -llapack -lblas -Wl,-rpath,<PSUADE.dir>/build/lib:
```

In the following we give examples on how to call a few PSUADE functions.

## 3.1 Calling Sampling Generation

A key function in any non-intrusive uncertainty quantification task involves creating a sample. This sub-section provides the code segment for creating a Latin hypercube sample. One first needs to set the number of inputs, number of outputs, and the sample size. Then, the input bounds are to be initialized (in the example below, the PSUADE vector class ('psVector') is used. This should be followed by instantiating the 'LHSampling' class, initializing the object ('setInputBounds', 'setOutputParams', and 'setSamplingParams' functions), and then the 'initialize' function can be called. At this point the generated sample resides inside the object and can be retrieved using the 'getSample' function (the 'vecSamOuts' and 'vecSamState' vectors are needed here as part of the general protocol).

```
....
#include "PsuadeInclude.h"
....
main()
{
   int    nInputs = 2, nOutputs = 1, nSamples = 10;
   psVector  vecLBs, vecUBs, vecSamInps, vecSamOuts;
   psIVector vecSamStats;
   vecLBs.setLength(nInputs);
   vecUBs.setLength(nInputs);
   vecLBs[0] = 0.009; vecUBs[0] = 0.011;
   vecLBs[1] = 0.225; vecUBs[1] = 0.275;

   LHSampling *sampler = (Sampling *) new LHSampling();
   sampler->setInputBounds(nInputs, vecLBs.getDVector(),
                           vecUBs.getVector());
   sampler->setOutputParams(nOutputs);
   sampler->setSamplingParams(nSamples, 1, 0);
   sampler->initialize(0);
   nSamples = sampler->getNumSamples();
   vecSamInps.setLength(nSamples * nInputs);
   vecSamOuts.setLength(nSamples*nOutputs);
   vecSamState.setLength(nSamples);
   sampler->getSamples(nSamples, nInputs, nOutputs,
```

```
                vecSamInps.getDVector(), vecSamOuts.getDVector(),
                vecSamState.getIVector());
   // Sample inputs are in vecSamInps as a vector
   ....
}
```

## 3.2 Calling Global Sensitivity Analysis

This sub-section presents the code segment for creating performing the first-order Sobol' analysis. To set up for the analysis, you need to instantiate and initialize the 'aData' object with the sample information. Subsequently, the 'analyze' function can be called with this object.

```
....
#include "PsuadeInclude.h"
....
main()
{
  ...
  aData anaDataObj;
  RSMSobol1Analyzer *analyzer = new RSMSobol1Analyzer();

  anaDataObj.nInputs_  = nInputs;
  anaDataObj.nOutputs_ = nOutputs;
  anaDataObj.nSamples_ = nSamples;
  anaDataObj.iLowerB_  = vecLBs.getDVector();
  anaDataObj.iUpperB_  = vecUBs.getDVector();
  anaDataObj.sampleInputs_  = vecSamInps.getDVector();
  anaDataObj.sampleOutputs_ = vecSamOuts.getDVector();
  anaDataObj.sampleStates_  = vecSamStats.getIVector();
  anaDataObj.printLevel_ = 3;
  analyzer->analyze(anaDataObj);
  // when these lines are run, sensitivity results will appear
  ...
}
```

## 3.3 Calling Optimizers

PSUADE has a few numerical optimization methods that can be linked into user programs. This sub-section describes the steps to do this linking on a Linux platform for a few of these optimizers.

### 3.3.1 Calling the Bobyqa Optimizer

Bobyqa, which was developed by Michael Powell, can solve bound-constrained optimization problems. To call this optimization function, do the following in the user program:

```
....
#include "PsuadeInclude.h"
....
/* define the user program here */
void userEvaluator(int nInputs, double *inputs, int nOuts, double *outputs)
{
   /* subroutine body - read from inputs and write to outputs */
   /* outputs[0] will store the objective function values */
}
...
main()
{
   int    ii, nInps=2, nOuts=3, maxEvals=1000;
   double tol=1e-6;
   psVector vecInputs;
   // instantiate initial guess vector and initialize
   vecInputs.setLength(nInps);
   ...
   // instantiate input lower and upper bounds vectors and initialize
   psVector vecLBs, vecUBs;
   vecLBs.setLength(nInps);
   vecUBs.setLength(nInps);
   ...
   // instantiate optimizer and initialize and optimize
   BobyqaOptimizer *opt = BobyqaOptimizer();
   opt->setObjectiveFunction(userEvaluator);
   opt->optimize(nInps, vecInputs.getDVector(), vecLBs.getDVector(),
                 vecUBs.getDVector(), nOuts, maxEvals, tol);
   double *X = opt->getOptimalX();
   for (ii = 0; ii < nInps; ii++)
      printf("Bobyqa final   X %d = %12.4e\n",ii+1,X[ii]);
   printf("Optimal Y = %e\n", opt->getOptimalY());
   printf("Total number of function evaluations = %d\n",
          opt->getNumFuncEvals());
   delete opt;
}
```

### 3.3.2 Calling the Cobyla Optimizer

Cobyla, which was developed by Michael Powell, can solve nonlinearly-constrained optimization problems. To call this optimization function, do the following in the user program:

```
....
#include "PsuadeInclude.h"
....
/* define the user program here */
void userEvaluator(int nInputs, double *inputs, int nOuts, double *outputs)
{
   /* subroutine body - read from inputs and write to outputs */
   /* outputs[0] will store the objective function values */
   /* outputs[1:nOuts-1] will store the constraint values */
   /*         constraint is feasible if it is <= 0)
}
...
main()
{
   int    ii, nInps=2, nOuts=3, maxEvals=1000;
   double tol=1e-6;
   psVector vecInputs;
   // instantiate initial guess vector and initialize
   vecInputs.setLength(nInps);
   ...
   // instantiate input lower and upper bounds vectors and initialize
   psVector vecLBs, vecUBs;
   vecLBs.setLength(nInps);
   vecUBs.setLength(nInps);
   ...
   // instantiate optimizer and initialize and optimize
   CobylaOptimizer *opt = CobylaOptimizer();
   opt->setObjectiveFunction(userEvaluator);
   opt->optimize(nInps, vecInputs.getDVector(), vecLBs.getDVector(),
                 vecUBs.getDVector(), nOuts, maxEvals, tol);
   double *X = opt->getOptimalX();
   for (ii = 0; ii < nInps; ii++)
      printf("Cobyla final   X %d = %12.4e\n",ii+1,X[ii]);
   printf("Optimal Y = %e\n", opt->getOptimalY());
   printf("Total number of function evaluations = %d\n",
          opt->getNumFuncEvals());
   delete opt;
}
```

### 3.3.3 Calling the SCE Optimizer

SCE (Nonsmooth Optimization by Mesh Adaptive Direct search), which was developed by Abramson, *et al*, can solve mixed integer and continuous variable optimization problems. To call this optimization function, you will first need to download the NOMAD library, compile it into PSUADE, and do the following in the user program:

```
....
#include "PsuadeInclude.h"
....
/* define the user program here */
void userEvaluator(int nInputs, double *inputs, int nOuts, double *outputs)
{
   /* subroutine body - read from inputs and write to outputs */
   /* outputs[0] will store the objective function values */
}
...
main()
{
   int    ii, nInps=2, nOuts=3, maxEvals=1000;
   double tol=1e-6;
   // instantiate initial guess vector and initialize
   vecInputs.setLength(nInps);
   ...
   // instantiate input lower and upper bounds vectors and initialize
   psVector vecLBs, vecUBs;
   vecLBs.setLength(nInps);
   vecUBs.setLength(nInps);
   ...
   // instantiate optimizer and initialize and optimize
   NomadOptimizer *opt = NomadOptimizer();
   opt->setObjectiveFunction(userEvaluator);
   opt->optimize(nInps, vecInputs.getDVector(), vecLBs.getDVector(),
                 vecUBs.getDVector(), nOuts, maxEvals, tol);
   double *X = opt->getOptimalX();
   for (ii = 0; ii < nInps; ii++)
      printf("Nomad final   X %d = %12.4e\n",ii+1,X[ii]);
   printf("Optimal Y = %e\n", opt->getOptimalY());
   printf("Total number of function evaluations = %d\n",
          opt->getNumFuncEvals());
   delete opt;
}
```