PSUADE Short Manual
(Version 1.7)

Charles Tong

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

Livermore, CA 94551-0808

May, 2017

Contents

1 Introduction

PSUADE (Problem Solving environment for Uncertainty Analysis and Design Exploration) is a software package for performing various uncertainty quantification (UQ) computational tasks such as uncertainty analysis (UA), sensitivity analysis (SA), parameter study, numerical optimization, Bayesian calibration, etc. It comprises three major components: a suite of sampling methods, a job execution environment, and a collection of analysis/optimization tools. This document describes how to set up and use these UQ tools. Detailed UQ mathematics can be found in the theory manual.

1.1 A Quick Start

Follow the instructions in this section and you should be able to build and run PSUADE (on a simple example) in less than 5 minutes (depending on the speed of your hardware) on a Linux-based system. For building PSUADE executables on other platforms (MAC, Windows), please refer to detailed instructions in a later section.

1. tar xvfz PSUADE_vx.x.x.tar

2. cd PSUADE_vx.x.x

3. For adding the external Gaussian process capability, please consult PSUADE developers.

4. Set the 'FC' environment variable to your preferred Fortran compiler, if you have one (cmake will select one automatically if not set).

5. mkdir build

6. cd build

7. ccmake .. (Select packages by typing 'c' and then using the arrow keys to move up and down the list and type 'enter' to select. When you are finished with package selection, type 'c' (may be twice) and then 'g' to save and exit. If you would like to install the executable somewhere else, set the install directory.)

8. make (to create the 'psuade' executable and libraries in the buiid/bin directory), or

9. make install (to install 'psuade' in your install directory if, for example, you would like to make the executable to be available to your project team).

10. To test correct installation, do:

    (a) cd PSUADE_vx.x.x/Examples/Bungee

    (b) cc -o simulator simulator.c -lm

    (c) psuade psuade.in (this is to verify that the executable runs correctly).

What you have just done are to build the PSUADE executable and perform uncertainty analysis on a simple example. At the end PSUADE prints out the summary statistics; and all sample input and output data will have been stored in the psuadeData file. Later on in this document more details about how to create PSUADE input files (psuade.in in this case) and how to create Matlab/Scilab graphics will be given.

1.2 PSUADE Capabilities

PSUADE supports non-intrusive uncertainty quantification through sampling (it does have a few features to support semi-intrusive methods). Some of the available sampling methods are:

- Monte Carlo (MC) and quasi-Monte Carlo (LPTAU),

- Latin hypercube (LH) and orthogonal arrays (OA, OALH),

- Morris one-at-a-time (MOAT), its variants, and other screening designs,

- Central composite designs (CCI4, CCI5, etc),

- Factorial (FACT) and fractional factorial (FF4, FF5)

- Fourier Amplitude Sampling Test (FAST)

- Other space-filling designs (e.g. METIS)

- Support for several standard input probability distribution functions (only for MC sampling)

- Some capabilities for adaptive sampling

Once these sample points have been generated, they are evaluated by running the user simulation codes with the sample values. PSUADE provides a mechanism to accomplish this via a runtime environment, which performs the following tasks when invoked (by, e.g. 'psuade psuade.in'): for each sample point,

- Write the values of a sample point to a parameter file (number of inputs in the first line, and input values in subsequent lines).

- Call the user simulation code (provided by users in the PSUADE input file - keyword is 'driver') with the parameter file as its first argument (and the name of the sample output file as its second argument).

- The user simulation code is expected to read in the parameter values from the parameter file, substitute the values in his/her application's input decks, run the simulation, extract target output quantities from simulation output files, and write them to the sample output file that has been specified as the second argument when PSUADE

4

calls the user program. Thus, the user program can be a simple program such as the compiled simulator.c in some of our examples, or a complex super-script performing pre-processing (inserting parameter values into the simulation), actual model evaluation (submitting and running simulation on the user-designated system), and postprocessing (extracting and writing simulation outputs).

- PSUADE detects the presence of the output file and reads in the simulation outputs of interest.

- PSUADE then continues with the next sample point until all sample points have been processed (Optionally, PSUADE can process multiple sample points at the same time using asynchronous or ensemble mode).

- Finally, PSUADE takes in all sample data (both inputs and outputs) and analyzes them based on user requests given in the PSUADE input file (the 'ANALYSYS' section).

PSUADE supports many types of analysis such as

- Parameter screening (several of these methods have been tailored for different types of simulation model and different scenarios)

- Response surface construction and validation (including adaptive response surface methods)

- Basic uncertainty and correlation analysis (for raw samples or response surfaces)

- Mixed aleatory-epistemic uncertainty analysis

- Global sensitivity analysis (first-order, second-order, and total-order Sobol' sensitivities),

- Bayesian calibration (with discrepancy modeling option)

- Hypothesis testing

- Deterministic numerical optimization (for various types such as with bound or inequality constraints, with derivatives, or with mixed continuous-integer variables)

- Optimization under uncertainty (for various types)

- Graphical analysis (e.g. scatter plots via Matlab/Scilab)

- Adaptive design of experiments

In addition, beginning with version 1.7.4, parallel processing has been added to some analysis methods (such as bootstrapped Sobol' analysis, some response surface methods, and Bayesian inference) for faster turnaround time. Finally, there are other advanced features in PSUADE which are under active research and are not described in this document.

2 Installation

In this section we describe installation procedures for three different operating systems.

2.1 Linux

As described in the last section, installation of PSUADE on Linux-based systems is straight-forward. After 'unzipping' and 'untarring' the downloaded file, go into the PSUADE directory and do the following:

```
[Linux] mkdir build
[Linux] cd build
[Linux] (optional) setenv FC <your preferred Fortan compiler>
[Linux] ccmake ..
    hit 'c'
    If you need to change the compiler, hit 't' and find the
       CMAKE_C_COMPILER and CXX fields and fix them.
    hit 'c'
    hit 'c' again until you are able to hit 'g'.
    hit 'g' to generate an exit

    * If you do not have ccmake, do :
    * cmake ..
[Linux] make
```

At the end of this installation, the PSUADE executable will have been created in the build/bin directory. Note that since 'BUILD_SHARED' is the default option, the executable will use the shared libraries in the build/lib directory, so it is important to keep the libraries at the same directory and be accessible by users. If it is desirable to have the executable and libraries accessible to multiple users, you can set the 'CMAKE_INSTALL_PREFIX' field in 'ccmake' and then issue 'make install' instead.

2.2 MacOSX

Building PSUADE executable from source files on MacOS is similar to building it on Linux systems. The major difference is that the MAC compilers may spit out more warnings, and possibly compiler errors that prevent a successful build. A build session is given below:

2.2.1 Step 1: Check Compilers

For this release the following compilers have been checked to ensure successful compilations.

```
[macos] cc --version (gcc also works)
Apple LLVM version 6.0 (clang-600.0.57) (based on LLVM 3.5svn)
```

```
Target: x86_64-apple-darwin13.4.0
Thread model: posix

[macos] c++ --version (g++ also works)
Apple LLVM version 6.0 (clang-600.0.57) (based on LLVM 3.5svn)
Target: x86_64-apple-darwin13.4.0
Thread model: posix

[macos] gfortran --version
GNU Fortran (GCC) 4.8.0 20120603 (experimental)
Copyright (C) 2012 Free Software Foundation, Inc.
```

2.2.2   Step 2: Run Cmake

```
[Linux] mkdir build
[Linux] cd build
[Linux] ccmake ..
    hit 'c'
    hit 't' to go to advanced options.
    I can see that in my case cmake has picked up cc:
    CMAKE_C_COMPILER                    /usr/bin/cc
    You can keep this the same, or you can change it to:
    CMAKE_C_COMPILER                    /usr/bin/gcc

    Once all compilers have been verified, hit 'c' until you can hit 'g',
    then hit 'g' to generate and exit.
```

2.2.3   Step 3: Build Executable

To build the PSUADE executable, do the following:

```
[Linux] make (or make install to another directory)
```

At the end of this installation, the PSUADE executable will have been created in the build/bin directory.

2.3   Windows

Building PSUADE executable from source files on Windows requires 'cmake', and 'mingw' (preferably including 'gfortran'). If you desire to build an installable package, you will need NSIS.

2.3.1   Step 1: Check Compilers

First make sure you have 'cmake' version 2.8 or higher installed on your system. Then,

```
Start the 'cmake-gui' program.
Select your PSUADE source tree, and where you want it to be built.
Click 'configure'.
Select MingGW make files.
Click 'Generate'.
```

2.3.2   Step 2: Build Executable

Open a command line window, either 'powershell' or 'cmd', then do:

```
cd builddir
c:\mingw\bin\mingw-make.exe (It should build for a while)
```

2.3.3   Step 3: Install

You can now install PSUADE by running

```
c:\mingw\bin\mingw-make.exe install
```

Now continue to read this manual and follow the instructions to get a simple application running.

3   Using PSUADE

PSUADE operates in one of the two modes: batch or command line modes.

3.1   Batch Mode

In batch mode, PSUADE interacts with users via a few files. At the first level, an PSUADE input file (named psuade.in below) has to be created and run via

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

This psuade.in file should normally begin with the keyword PSUADE as the first line and should have 5 subsections (the ANALYSIS section can be omitted) with the last line having the keyword END. The formats of the subsections are described next (for an example, read the psuade.in file in the Examples/Bungee directory).

3.1.1 The Input Section

The INPUT section allows the users to specify the number of inputs, their names, their ranges, and their probability distributions. Specifically, it is enclosed in an **INPUT** block. An example is given as follows:

```
INPUT
   dimension = 4
   variable 1 X1  =   0.0    1.0
   variable 2 X2  =   0.0    1.0
   variable 3 X3  =   0.0    1.0
   variable 4 X4  =   0.0    1.0
   PDF 1 T  0.0    1.0
   PDF 2 N  0.0    1.0
   PDF 3 S  sample3 1
   PDF 4 N  0.0    1.0
   COR 2 4  0.5
END
```

In this example the number of inputs is 4, their names are X1, X2, X3, and X4 (notice that the variable indices are 1-based), and their lower and upper bounds are all 0 and 1, respectively. The probability distributions (PDF) for the inputs are optional (the default is uniform U with lower and upper bounds specified in the variable declaration). If the PDF is either normal (N) or lognormal (L), the mean and standard deviation must also be provided. If the PDF is triangular (T), the mean and width must be provided. Other available distributions are beta (B), exponential (E), gamma (G), Weibull (W), and user-provided samples (S). The last option (S) allows probabilities to be represented by a pre-generated sample file ('sample3' above) such as a posterior samples generated from Bayesian inference (the integer following the sample file is the index specifying which column in the sample file is to be used for 'X3'). An example of the 'S' type sample file can be found in Examples/PDFTest ('sample1' or 'sample1'). The 'S' option also facilitates the use of discrete variables even though PSUADE assumes all variables to be continuous. 'COR' specifies the correlation between two inputs and it is currently available only for the normal distribution (that is, for correlation to work in PSUADE, both input 2 and 4 have to be of PDF type 'N').

3.1.2 The Output Section

The OUTPUT section is similar to but simpler than the INPUT section. Here only the output dimension and the names of the output variables to be specified. For example, given as follows:

```
OUTPUT
   dimension = 3
   variable 1 Y1
   variable 2 Y2
```

```
      variable 3 Y3
   END
```

3.1.3   The Method Section

The METHOD section specifies the selected sampling method and additional information on sampling. An example is given below.

```
METHOD
   sampling = LH
   num_samples = 600
   num_replications = 60
   num_refinements = 0
   randomize
   random_seed = 129932931
END
```

In this example, the sampling method is Latin hypercube, the sample size has been set to 600, and no refinement is used (refinement is an advanced feature for adaptive sampling and is described in detail in the theory manual). When the number of replications is larger than 1, it is called replicated Latin hypercube which is useful for certain global sensitivity analysis. In this example, with 60 replications, the number of levels for the Latin hypercube samples is $600/60 = 10$. Also, the randomize flag has been turned on to tell the sampling method that random perturbation should be added to the sample. Optionally, the random number generator seed can be provided. This is useful if you would like your sampling experiments to be repeatable (having the same random seed every time the experiment is repeated.)

Some of the other sampling methods available in PSUADE are (refer to the theory manual or examine your sample data file, psuadeData, for more details):

```
MC    - Monte Carlo
LPTAU - a quasi-random sequence
FACT  - full factorial design
MOAT  - Morris one-at-a-time screening
LH    - Latin hypercube
OA    - Orthogonal Array
OALH  - Orthogonal Array-based Latin hypercube
FAST  - Fourier Amplitude Sampling Test (FAST)
BBD   - Box Behnken design
PBD   - Plackett Burman design
FF4   - Fractional factorial of resolution IV
FF5   - Fractional factorial of resolution V
FF5   - Fractional factorial of resolution V
CCI4  - Central composite (circumscribed) of resolution V
CCI5  - Central composite (circumscribed) of resolution V
```

```
    METIS - full space-filling based on domain decomposition
    SPARSEGRID - a sparse grid method
```

3.1.4 The Application Section

The APPLICATION section sets up the user-provided simulation executable and other runtime parameters. An example is given below.

```
    APPLICATION
        driver = ./testmain
        opt_driver = NONE
        aux_opt_driver = NONE
        ensemble_run_mode
        ensemble_driver = NONE
        ensemble_opt_driver = NONE
        max_parallel_jobs = 1
        max_job_wait_time = 1000000
    END
```

Here driver points to the executable to be used as the simulation model, and opt_driver and aux_opt_driver point to the executables for numerical optimization (aux_opt_driver is needed for the 'SM' and 'MM' optimizers, which are two-level optimizers), if they are needed. Again, the user code can just be a simple program or a complex super-script performing preprocessing, actual model evaluation, and postprocessing. The user code can also be a PSUADE sample data file, as will be shown later.

After the creation of a sample based on information from the INPUT, OUTPUT and METHOD sections, PSUADE proceeds with launching the jobs. If the keyword max_parallel_jobs is set to 1, the sequential mode is turned on. In this mode, PSUADE schedules the evaluation of the user-provided function by sequencing from sample point 1 onward. To run job i, PSUADE first creates an input parameter file (called psuadeApps.in.i). This file contains in its first line the input dimension, followed by the values of the input parameters for the i-th sample point. PSUADE then calls driver with two parameters (for example, for the sample point 9)

```
        ./testmain psuadeApps.in.9 psuadeApps.out.9
```

The driver program is expected to take the input parameters from the psuadeApps.in.9 file, do whatever is needed, and write the outputs to the psuadeApps.out.9 file. An example of the content of an input file (2 variables) created by PSUADE is:

```
        2
        0.345
        1.429
```

An example of the content of an output file (e.g. 3 output variables) to be created by user programs is:

```
    3.12
    15.9
    100.4
```

If max_parallel_jobs is set to a number larger than 1, the asynchronous job scheduling mode will be turned on. In this mode, multiple psuadeApps.in.i files are created simultaneously, and driver is called max_parallel_jobs times simultaneously. max_job_wait_time is used for fault detection and recovery.

For fast simulations (such as 'simulator.c' in Examples/Bungee), cycling through the sample points one by one will require file input/output (open the parameter file, read in the sample point, evaluate, and write to the output file) that may consume much more time than the calculations themselves. To reduce the overall processing time, PSUADE provides the ensemble_driver (and ensemble_opt_driver for numerical optimization) that can be called in place of driver (and opt_driver) to facilitate 'group processing' when 'ensemble_run' mode is turned on. In this mode, PSUADE writes multiple sample points (as specified by the max_parallel_jobs variable) in the parameter file and the user executable is expected to process all sample points before returning the results to PSUADE. An example is given in the Examples/Bungee directory (uncomment the 'ensemble' lines in 'psuade.in', compile ensemble_simulator.c and run 'psuade psuade.in').

Some of the other options in this section are:

```
launch_only          - launch all jobs without waiting for results
gen_inputfile_only   - generate all sample files only (no code runs)
limited_launch_only  - launch max_parallel_jobs jobs and terminate
```

launch_only is useful when you can run all the jobs at the same time on your machine but each job takes a long time to run. In this case PSUADE is terminated after all jobs have been launched, and when all the jobs are completed, PSUADE should be launched again to harvest the results.

gen_inputfile_only is useful when job launching management is handled outside PSUADE. In this case PSUADE is launched initially to create the input files for all sample points (all psuadeApps.in.i). At this point your preferred job scheduler can take over to run all sample points. After the runs have been completed and all psuadeApps.out.i have been created, PSUADE should be called again to harvest the results.

Finally, limited_launch_only is useful in a semi-automated job launching environment (e.g. on the LC machines at LLNL). In this case PSUADE is called initially to launch a small number of jobs (e.g. 10 out of 100 in total). The job execution scripts for these 10 jobs are equipped with capabilities to launch more jobs when it is completed (e.g. the dependency auto-submission mechanism on the LC machines). This 'domino effect' job launching (also called 'chain mode') will continue until all 100 jobs have been completed. After that, PSUADE should be called again to harvest the results.

3.1.5 The Analysis Section

The ANALYSIS section specifies the type of analysis to be performed and the parameters to be used. An example given below ('analyzer method = Moment') computes statistical moments on output number 1 ('analyzer output_id = 1'). Lines beginning with '#' are treated by PSUADE as comment lines (the commented options will be explained later).

```
ANALYSIS
   analyzer method = Moment
   analyzer threshold = 5.000000e-04
   analyzer output_id  = 1
   #analyzer rstype = MARS
   #optimization method = bobyqa
   #optimization num_local_minima = 1
   #optimization use_response_surface
   #optimization num_fmin = 1
   #optimization fmin = 0
END
```

Some of the available analysis methods are:

```
MainEffect      - sensitivity indices
TwoParamEffect - second order sensitivity indices
RSFA            - response surface (RS) analysis (curve fitting)
MOAT            - Morris one-at-a-time screening analysis
Correlation     - correlation analysis
Integration     - numerical integration using the data points
FAST            - Fourier Amplitude Sampling Test analysis
FF              - fractional factorial main and interaction analyses
PCA             - principal component analysis
RSMSobol1       - RS-based first order sensitivity analysis
RSMSobol2       - RS-based first second sensitivity analysis
RSMSobolG       - RS-based group main effect analysis
RSMSobolTSI     - RS-based total sensitivity analysis
```

When response surfaces are used together with the selected analysis method, the response surface type ('rstype' above) may have to be specified. Some of the available response surface types are (some of these are not included in the releases):

```
MARS            - multi-variate adaptive splines (by Friedman)
MARSBag         - MARS with bootstrapped aggregation
linear          - linear regression
quadratic       - second-order regression
cubic           - third-order regression
quartic         - fourth-order regression
```

```
user_regression    - regression with user-specified basis functions
GP1                - Gaussian process (TPROS)
GP2                - Gaussian process (PSUADE's local implementation)
SVM                - support vector machine
Kriging            - an universal Kriging method
SOT                - a sum-of-trees method
KNN                - K nearest-neighbors
RBF                - Radial Basis functions (with additional options)
splines            - 2D or 3D splines (on factorial designs)
selective_regression - user-specified subset of polynomials
sparse_grid_regression
```

In addition, for performing numerical optimization, a few related options have to be specified (the commented lines from the above example). In the example, the selected optimization method is bobyqa by Michael Powell. Other available optimization methods are crude (a simple examination of all sample outputs and select the minimum one), minpack (an external optimization package), and sm/mm (space-mapping and manifold-mapping methods by David E. Ciaurri). num_local_minima tells PSUADE how many minima to identify (from the initial sample) for multi-start searches. If user_response_surface is used, the sample data will first be used to create a response surface before searching for minima in the initial sample. num_fmin tells PSUADE the number of optimal points to be expected so that PSUADE can pick the best optimal points (num_fmin needs to be smaller or equal to num_local_minima. Users can also tell PSUADE the optimal value to look for via fmin.

3.2   Command Line Mode

PSUADE allows users to 'interactively' perform some of the analyses. The idea is to run all the simulations with the batch mode (and delete or comment out all analysis methods in the input file). Once all simulations have been completed, the psuadeData file will contain all sample inputs and the corresponding outputs enclosed in the PSUADE_IO section. This file (which needs to be renamed) is to be loaded in the command line mode for analysis. Command line mode is launched by calling

```
[Linux] psuade
```

without any argument. Some of the available commands in the command line mode are (most of the commands can be found by the help command and also in the reference manual):

```
load <filename>  (load a data file, e.g. psuadeData)
splot            (generate scatter plot in matlab)
moat             (Morris analysis on Morris samples)
me               (main effect study + matlab plot)
rs2              (2-input response surface in Matlab)
rs3              (3-input response surface in Matlab)
```

14

```
rscheck            (Check quality of response surfaces)
rssobol1b          (Perform first order Sobol' analysis)
quit
help
```

For example, after you have completed a set of runs, a PSUADE data file will have been created (say, the renamed file is psData). To create scatter plots for the data in the command line mode, do:

```
[Linux] psuade
*** ************************************************************ ***
*** Welcome to PSUADE (version 1.7.8)                          ***
*** ************************************************************ ***
PSUADE - A Problem Solving environment for
Uncertainty Analysis and Design Exploration
psuade> load psData
psuade> splot
matlabsp.m is now available for scatter plots.
psuade> quit
[Linux]
```

You can now use Matlab to display the scatter plot. You can also generate Scilab files by toggle the 'scilab' command (Scilab capability is less well-developed than Matlab capability in PSUADE).

The command line mode contains many commands not only for statistical analysis, but also for sample manipulation and visualization. Use 'help' to explore the rich collection of commands.

4   Examples

PSUADE provides many tools for answering UQ questions concerning the simulation model. For example, given a computational model simulating some physical processes,

1. What is the impact of parameter uncertainties on the model output of interest? (uncertainty analysis)

2. Is there a small subset of parameters accounting for most of the output variabilities? If so, what is this subset? (parameter screening/down-select)

3. How to construct a relationship between some input parameters and the model outputs of interest? (response surface modeling)

4. How much of the output uncertainties can be attributed to individual parameters? (global sensitivity analysis)

5. What are the parameter values that give the best model performance? (optimization)

6. What are the parameter values that give the best model performance when my simulation model has uncertainties? (optimization under uncertainty)

7. What are the parameter values that best fit the available experimental data in view of uncertainties? (Bayesian calibration, parameter estimation)

8. What physical experiment(s) should be performed next to minimize prediction uncertainties? (Optimal design of experiments)

9. How to visualize uncertainty data (for purposes of identifying outliers, observing trends, etc.)?

In the following we provide a few examples to show in more details how to set up and run PSUADE. PSUADE has many other advanced features for handling complex multi-physics models.

4.1 Uncertainty Analysis

This section shows how to perform a simple uncertainty analysis on the following Rosenbrock function: let the function be given by

$$Y = \sum_{i=1}^{m-1} (1 - X_i)^2 + 100(X_{i+1} - X_i^2)^2 \quad X_i \in [0, 2]$$

where m is the number of input parameters and can be any integer larger than 1 (we use $m = 6$ in this example). To compute the basic statistical moments of this function assuming all inputs are uniformly distributed in $[0, 2]$, we select the Latin hypercube design with a sample size of 100 (an arbitrary pick). The simulations are to be run in sequential mode. The corresponding PSUADE input file (say, 'psuadeRawUA.in') is:

```
PSUADE
INPUT
   dimension = 6
   variable 1 X1   =    0.0    2.0
   variable 2 X2   =    0.0    2.0
   variable 3 X3   =    0.0    2.0
   variable 4 X4   =    0.0    2.0
   variable 5 X5   =    0.0    2.0
   variable 6 X6   =    0.0    2.0
END
OUTPUT
   dimension = 1
   variable 1 Y
```

```
END
METHOD
   sampling = LH
   num_samples = 100
   randomize
   random_seed = 128121211
END
APPLICATION
   driver = ./testmain.py
END
ANALYSIS
   analyzer method = Moment
   printlevel 3
END
END
```

Here 'randomize' in the 'METHOD' section specifies that random perturbations are to be added to the sample; and 'random_seed' is defined so that repeating this exercise will produce the same result (since perturbation added to the Latin hypercube sample depends on the seed of the random number generator). The 'driver' points to an executable, which takes a sample point from a parameter file (first argument), uses it to run a simulation, and writes the simulation output to the output file (second argument). The driver program can be in any language provided that it can be executed (by the Linux 'system' command). Our example uses Python to represent the above function (Examples/UserExample/testmain.py):

```
#!/usr/local/bin/python
import string
import sys
infile  = open(sys.argv[1], "r")
lineIn  = infile.readline()
ncols   = lineIn.split()
n       = eval(ncols[0])
X       = range(n)
for ii in range(n):
   lineIn  = infile.readline()
   ncols   = lineIn.split()
   X[ii]   = eval(ncols[0])
infile.close()
Y = 0
for ii in range(n-1):
   Y = Y + pow(1 - X[ii], 2) + 100 * pow(X[ii+1] - X[ii] * X[ii], 2)
outfile = open(sys.argv[2], "w")
outfile.write("%e \n" % Y)
outfile.close()
```

After these files have been prepared (make sure the Python link in **testmain.py** is correct, and that **testmain.py** has 'execute' permission turned on), run PSUADE:

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

and, at the completion of the runs, the moment information will be displayed and the **psuadeData** file (which contains the 100 sets of sample inputs/outputs) will also be available for use in further analysis.

The above shows how to perform uncertainty analysis in batch mode. This analysis can also be conducted in the following three steps:

1. Run the simulation and ignore the analysis (by commenting out the 'analyzer method' line in **psuadeRawUA.in**);

2. Rename **psuadeData** to, for example, **psData**);

3. Load **psData** in command line mode and run the **ua** command to see the sample statistics; and

4. Optionally, launch Matlab and run 'matlabua.m' to visualize the output probability distribution.

This three-step approach is recommended because the command line interpreter provides many other functions to analyze the same data set (e.g. **ca** for correlation analysis).

In this example, we observe that even for computing simple statistical moments, a sample size of 100 may be too small. You may increase the sample size and re-run to see if the result changes much. You can also turn on the refinement mode (by uncommenting the line 'num_refinements = 5' and re-run), which will iteratively increase the sample size and re-anaylze. An even more efficient (response surface-based) method will be discussed later.

4.2   Screening for Important Inputs

A useful design and analysis tool in PSUADE is the suite of parameter screening methods. Parameter screening or down-selection can be useful when the number of uncertain parameters is large (say, > 10) because many UQ analyses suffer from the 'curse of dimensionality', which often results in prohibitively high computational cost and inaccurate analysis results. The objective is to use relatively inexpensive methods to quickly identify the most important parameters for subsequent more computationally intensive analysis (e.g. response surface analysis, variance-based sensitivity analysis).

There are several parameter screening methods that may be useful under different scenarios. In this section we provide a survey of these methods and their applicability; and then we will show how to use one of them.

If you already have a sample (somewhat space filling) available for analysis (that is, simulations are complete), the following screening methods may be useful:

1. Correlation analysis (ca). This method, however, assumes that the model input-output relationship is more or less linear. If the model input-input relationship is not linear but exhibits monotonic behavior (non-decreasing or non-increasing), the Spearman coefficients are more informative. For general nonlinear relationships, this analysis may give erroneous results.

2. Delta test (delta_test). This method works only with relatively large (hundreds to a thousand) random or quasi-random samples. Since this method involves minimization of certain noise function, it may be computationally intensive. Screening results may be confirmed by using different sample sizes. This method is not suitable for very large parameter set (e.g. > 50).

3. Sum-of-trees method (sot). This method works well with relatively large random or quasi-random samples (thousands or more). It uses bisection techniques to form unbalanced trees and estimates sensitivities based on frequencies of bisection in each input parameter.

4. Approximate response surface methods. If your already-completed sample is small, you can perform a response surface analysis on your sample and use the response surface to rank parameters. The two methods available in this class are mars_sa (based on the MARS response surface method) and gp_sa (based on Kriging). To assess whether these methods may be useful, we recommend first analyzing the response surface cross validation errors (described in next section) to make sure the response surface gives the right trend (error distribution centers around zero). These methods differ from the response surface analysis (to be discussed next) is that high accuracy is not a major concern (the objective is to give rough estimates based on trends).

If you do not have a sample ready for analysis, and the simulations are computationally expensive, a more careful sample design is needed. Again, design selection depends on your knowledge about the simulation model and how much computational resources are available. PSUADE currently provides the following screening methods:

1. If the model input-output is known to have a near-linear relationship, then the Plackett-Burman or local sensitivity analysis design (and the corresponding lsa analysis command) may suffice. These method requires only $m + 1$ simulations (m is the number of uncertain parameters).

2. If, in addition to linear relationship between each input and the model output, there are also interaction terms (e.g. the model equation consists also of terms involving two or more inputs, then the fractional factorial designs and the corresponding analysis (ff) may be useful.

3. For general non-parametric models that may be nonlinear with significant input parameter interactions (higher order sensitivities), we recommend the Morris (MOAT) method, which is an effective variable selection method when the number of inputs is large (say, $10 - 100$'s).

In the following we show how to use PSUADE to set up the **MOAT** screening analysis. The PSUADE input file for a 20-dimension problem is given in the Examples/Morris20 directory):

```
PSUADE
INPUT
   dimension = 20
   variable  1 X1  =   0.0   1.0
   variable  2 X2  =   0.0   1.0
   ...
   variable 20 X20 =   0.0   1.0
END
OUTPUT
   dimension = 1
   variable 1 Y
END
METHOD
   sampling = MOAT
   num_samples = 210
   randomize
END
APPLICATION
   driver = ./simulator
END
ANALYSIS
   analyzer method = MOAT
   printlevel 3
END
END
```

Here the sample size should be a multiple (usually 10) of $m + 1$ where m is the number of inputs. The driver program can be constructed in a similar manner as before (and thus is not to be given here). Again, PSUADE is launched with this input file and screening results will be displayed at completion.

Alternatively, the analysis can be performed interactively by (again **psuadeData** has been created and has been renamed to **psData**):

```
[Linux] psuade
*** ********************************************************** ***
*** Welcome to PSUADE (version 1.7.8)                         ***
*** ********************************************************** ***
PSUADE - A Problem Solving environment for
Uncertainty Analysis and Design Exploration
```

```
psuade> load psData
load complete : nSamples = 210
nInputs  = 20
nOutputs = 1
psuade> moat
... (MOAT results) ...
...
Create screening diagram ? (y or n) y
matlab/scilab screening diagram file name (no extension): screen
MOAT screening diagram matlab file = screen.m
psuade> quit
[Linux]
```

Thereafter, you can launch Matlab and run **screen.m** to view the Morris screening diagram (scatter and bootstrap plots are also available for more detailed analysis of anomalies and trends). The final decision on which parameters to be considered as important should be based on expert judgment (by modellers who know the simulation physics well, with the help from analysts who know how to interpret the MOAT results).

4.3 Response Surface Analysis

As we can see from the previous exercise, even a relatively simple test function may require large samples to compute the statistics accurately. To reduce the computational demands, one approach is to generate a small sample for this function and find out if the model input-output relationship can be described by a simple surface fitting method. In this example we show how to use the surface fitting tools in PSUADE to find the best response surface (or surrogate) model. To do this, we run PSUADE on Examples/UserExample/psuadeRS.in to evaluate a sample of size 100. We then rename the sample file **psuadeData** to **psData** so that it will not be overwritten by PSUADE (since 'psuadeData' is the PSUADE default output file). The following gives a snapshot of how to perform response surface analysis in the PSUADE command line mode:

```
[Linux] psuade
**********************************************************************
*       Welcome to PSUADE (version 1.7.8)
**********************************************************************
PSUADE - A Problem Solving environment for
         Uncertainty Analysis and Design Exploration
(for help, enter <help>)
======================================================================
psuade> load psData
load complete : nSamples = 100
nInputs  = 6
```

```
    nOutputs = 1
    psuade> rscheck (NOTE: this command performs response surface analysis)
    ... list response surface types ....
    Enter you choice? 2 ('2' is for quadratic regression)
    ... (some analysis results show be shown)
    Perform crosss validation ? (y or n) y
    Enter the number of groups to validate : (2 - 100) 20 (Use 10 - 20)
    ... (more informatin will be displayed)
    Random selection of leave-out groups ? (y or n) y (generally say yes)
    ...
    RSA: final CV error = -3.694e+02 (avg unscaled)
    RSA: final CV error = -5.043e-01 (avg   scaled)
    RSA: final CV error =  4.157e+02 (rms unscaled)
    RSA: final CV error =  6.512e-01 (rms   scaled)
    RSA: final CV error =  1.061e+03 (max unscaled, BASE=2.432e+03)
    RSF: final CV error =  2.879e+00 (max   scaled, BASE=2.479e+02)
    ...
    CV error file is RSFA_CV_err.m
    **************************************************************************
    psuade> quit
    [Linux]
```

After loading the sample, the 'rscheck' command is used to launch response surface analysis. Questions will be prompted on which response surface method to use and on which output (if there is more than one), additional information for different method when the response surface expert mode has been turned on, and whether to perform cross validation (CV) and how many CV groups to use. The default number of CV group is 10, but users are encouraged to experiment with different number of groups to enhance robustness.

Following the procedure and upon exit from PSUADE, there should be a file called 'RSFA_CV_err.m' in your working directory. This file should be run in Matlab, giving two plots: the left showing the distribution of the cross validation errors and the right showing qualitatively how good the selected curve fitting method is. In addition, when the response 'analysis' expert mode has been turned on before running 'rscheck', another file called 'RSFA_training_error.m' will be created. To assess, the 'goodness' of the response surface, we recommend examining the following quantities:

1. The 'final CV errors' similar to those shown above give important information. For example, if there is a large departure of the scaled average error from zero, a significant systematic bias may be present. If the scaled maximum error is large (> 1) with non-negligible base (e.g. if BASE=0.53 and max scale error is 0.5, it means the maximum error has been found to be $0.53 * 0.5 = 0.265$ when the output is 0.53), the fitting error may be significant.

2. Launching Matlab and running 'RSFA_training_error.m' will give a few plots. The Matlab Figure 2 will show the error distribution when the response surface is evaluated at the training set (also called resubstitution test). If there are significant errors in this step, the response surface may be declared 'unfit' and no further analysis should be performed. The left plot in Matlab Figure 1 shows the 'training errors' for individual sample points and the right plot compares the actual sample data against the predicted values from the response surface (perfect predictions put every point on the diagonal line). The Matlab Figure 3 displays interpolation errors with respect to each input and is useful in assessing which input parameters may need more attention.

3. The left plot in Figure 1 of 'RSFA_CV_err.m' gives the distribution of the cross validation errors. The desirable distribution should be centered around zero with small spread. Again, if the center is far away from zero, it indicates possible systematic bias.

4. The right plot in Figure 1 of 'RSFA_CV_err.m' compares the CV predictions with the actual simulation data. Ideally all points should lie on the diagonal line. For this example problem, since the function is a fourth-order polynomial and you use quadratic polynomials, significant CV errors showing systematic bias should be observed.

5. You can also change the 'morePlots' variable in 'RSFA_CV_err.m' to 1 and re-run in Matlab to create another plot showing the normalized CV errors (with respect to the output values). In our example problem, we observe that when quadratic regression is used, the result response surface incurs larger errors at the low end of the output range.

6. One more caveat: sometimes the cross validation errors may appear to be small, but the small errors may be deceiving. For example, if the sample outputs vary between 100 and 101, a maximum-scaled CV error of 0.001 gives an absolute error of ≈ 0.1, which may not be significant compared to 100, but may be significant when we notice that the output range is small (1.0).

After some preliminary analysis, it should be clear that fitting with quadratic regression does not lead to a satisfactory response surface. Since this is a fourth order polynomial, analyzing the data set with cubic regression will not improve the quality of the response surface either. However, the data set of size 100 is not large enough to analyze with quartic regression (which needs a sample size of 210). To see that quartic regression is the ideal candidate, you can change the 'num_samples' field in psuadeRS.in to 220 (210 plus a few more to allow room for cross validation) and run the simulations again. After that, run 'rscheck' with quartic regression and cross validation with 22 groups and observe an almost perfect fit.

You may experiment with other response surfaces and compare their error properties. If you are not satisfied with all available response surfaces, you may

1. Experiment with your own basis functions using the user-defined regression option. In this case, the response surface function is expected to be in the form

$$Y = \sum_{i=1}^{n} a_i \phi_i(X).$$

where X is the set of uncertain parameters and a_i's are the coefficients to be determined using regression techniques within PSUADE. To use this option you need to provide PSUADE with the following information:

   (a) the number of terms n, and

   (b) an executable file that returns the values of all the terms $\phi_i(X)$ given X for all points requested by PSUADE.

2. Add more sample point via sample refinement until the response surface can be 'validated' (using the refine or a_refine functions).

Once you have identified a suitable response surface, it will be ready for subsequent analysis. Note that there are two types of response surfaces provided by PSUADE - the ones that predict only the output given the input values (splines, MARS, sparse grid, SVM), and the ones that also provide errors associated with the predictions (polynomial and user-defined regressions, MARS with bootstrapping, Kriging, nearest-neighbors, sum-of-trees), so if you desire to include response surface uncertainties into subsequent analysis, select the ones that provide errors.

Once a suitable response surface has been constructed, many users express interest in getting the actual stand-alone code for future interpolation using this response surface. Users can access this option in the new PSUADE version (1.7.2 and after) via turning on the 'rs_codegen' mode in command line mode *before* running 'rscheck'. When the operation is complete, a file called 'psuade_rs.info' containing the 'C' or 'C++' interpolation code will have been created. In addition, a file called 'psuade_rs.py' containing the Python interpolation code will also have been created.

4.4   Response Surface-Based Uncertainty Analysis

To perform uncertainty analysis on a response surface with PSUADE, one should use the rsuab command. The steps are:

1. Generate a sample to run through the validated response surface.

   (a) Modify the INPUT section of your PSUADE sample that has been used to create a response surface (e.g. change ranges or add distributions to reflect desired input distributions).

   (b) Start PSUADE in command line mode, load the modified file, and use the 'gensample' command to create a large sample (say, of size 50000). Write this large into another file, say, ua_sample.

2. Launch PSUADE in command line mode to run uncertainty analysis:

   (a) Load psData,

   (b) Run the rsuab command,

   (c) Enter ua_sample when prompted for a sample file, and

   (d) When completed, a file called 'matlabrsuab.m' will have been created.

3. Launch Matlab and run 'matlabrsuab'.

   (a) The top left plot gives the model output probability distribution,

   (b) The bottom left plot gives the model output probability distributions for each bootstrap,

   (c) The right plot gives the corresponding cumulative probability distributions, and

   (d) If the response surface uncertainty is significant, the bootstrapped probability distribution curves will be easily distinguished.

Another command for response surface-based uncertainty analysis is the 'rsua' command, which performs an average case or worst case analysis. Use 'rsua -help' to see what this function does.

4.5   Quantitative Sensitivity Analysis

Quantitative sensitivity analysis include main effect, pairwise interaction effect, group main effect, and total sensitivity analysis. Since quantitative sensitivity analysis requires a large sample, it is often performed on validated response surfaces, unless simulations are inexpensive or a large sample of unknown sampling design is already available.

Main effect analysis studies the first order sensitivities of individual input parameter based on variance decomposition. Sensitivity (Sobol') indices can be computed using one of the following four methods:

1. Use the command 'me' (main effect) on the sample if a large sample (thousands or more) is already available;

2. Use replicated LH directly on the simulator or its response surface;

3. Use FAST sampling directly on the simulator or its response surface; or

4. Use direct numerical integration on the response surface.

In the following example, we describe the use of the replicated Latin hypercube approach directly on the simulator. The input file is given as follow (this can be found in the Examples/UserExample/psuadeME.in file):

```
PSUADE
INPUT
   dimension = 6
   variable  1 X1  =   0.0   2.0
   variable  2 X2  =   0.0   2.0
   variable  3 X3  =   0.0   2.0
   variable  4 X4  =   0.0   2.0
   variable  5 X5  =   0.0   2.0
   variable  6 X6  =   0.0   2.0
END
OUTPUT
   dimension = 1
   variable 1 Y
END
METHOD
   sampling = LH
   num_samples = 1000
   num_replications = 50
   randomize
END
APPLICATION
   driver = ./testmain.py
END
ANALYSIS
   analyzer method = MainEffect
END
END
```

Here the sample size is 1000 based on 50 replications of Latin hypercube each with $1000/50 = 20$ levels. To run this analysis, go to Examples/UserExample and run the following (first make sure to change the file permission to allow Python script to be execute-ready):

```
Linux] psuade psuadeME.in
```

At the conclusion of the analysis, main effect statistics will be displayed. More information will be displayed if the printlevel level is increased. In addition, a Matlab plot of the main effects (matlabme.m) will have been created.

Since a sample of size 1000 may not be sufficient to give reasonable results, we describe in the next example the use of the replicated Latin hypercube on response surfaces. First, run PSUADE on Examples/UserExample/psuadeRS.in to create a sample of size 100 for use in constructing a response surface. Again, rename the file psuadeData to psData and then run PSUADE again with Examples/UserExample/psuadeRSME.in, which is given below (It is important that you set the response surface type in psData before you launch this run).

```
PSUADE
INPUT
   dimension = 6
   variable 1 X1  =   0.0   2.0
   variable 2 X2  =   0.0   2.0
   variable 3 X3  =   0.0   2.0
   variable 4 X4  =   0.0   2.0
   variable 5 X5  =   0.0   2.0
   variable 6 X6  =   0.0   2.0
END
OUTPUT
   dimension = 1
   variable 1 Y1
END
METHOD
   sampling = LH
   num_samples = 50000
   num_replications = 50
   randomize
END
APPLICATION
   driver = psData
END
ANALYSIS
   analyzer method = MainEffect
END
END
```

Again, simple run the following command

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

and main effect results will be displayed, along with a few Matlab plot files (if selected). Specifically, the scatter plots show how the output behaves with respect to each individual input and the bootstrapped plot include errors with each main effect.

Again, one can use command line interpreter to perform the main effect analysis (by running the above script without setting the analysis method, loading the result file, and use the me command). Note that me will work for any sampling design (not just replicated LH) although the result will be less informative.

PSUADE also provides the functionality to perform second order (actually first and second order) sensitivity analysis on raw sample data (that is, not response surface evaluations) using ie in command line mode. In this case, instead of using replicated Latin hypercube, replicated orthogonal array design will be more appropriate although any space-filling sampling design will work. To achieve sufficient accuracy, however, very large sample is needed,

and hence it makes more sense to use response surfaces. To experiment with this analysis, you can run PSUADE with Examples/UserExample/psuadeRSIE.in. After completing this run, you can also load the sample data and have additional analysis.

PSUADE also provides the tsi command to perform total sensitivity analysis on a given sample. This sample needs to be even larger and with small number of parameters ($<= 21$) to give meaningful results. To experiment with this function, run PSUADE on Examples/UserExample/psuadeRSTSI.in to produce the psuadeData file and rename it. Launch the PSUADE command line mode, load the data file, and run 'tsi'.

The final example in this section shows how to compute sensitivity information in an alternative way - using direct numerical integration. Once the response surface has been validated and deemed satisfactory, it psData can be loaded into PSUADE's command line interpreter (make sure you indicate which response surface to use inside psData before loading the sample file):

```
[Linux] psuade
*** ********************************************************** ***
*** Welcome to PSUADE (version 1.7.8)                        ***
*** ********************************************************** ***
PSUADE - A Problem Solving environment for
         Uncertainty Analysis and Design Exploration
psuade> load psData
load complete : nSamples = 100
               nInputs  = 6
               nOutputs = 1
psuade> rssobol1b
...
Choose which output
Choose how many bootstrap samples to use
...
...
rssobol1 Statistics (based on 100 replications):
Input    1: mean =   1.1243143e+00, std =   0.0123132e+00
Input    2: mean =   2.5523545e+00, std =   0.1232000e+00
Matlab plot for first order sensitivities is in matlabrssobol1b.m.
psuade> quit
[Linux]
```

At the conclusion of the session, the main effects together with their standard deviations will be displayed. In addition, a Matlab file will also be available for visualizing the main effects. This analysis is different from 'me' on response surfaces in that it provides error bars to each sensitivity index based on bootstrapping. You can mimic this function with 'me' by generating multiple bootstraps yourself, running each with 'me', and computing the means and standard deviations of the sensitivity indices. Similarly, PSUADE provides

'rssobol2b' and 'rssoboltsib' to compute second and total order sensitivities with error bars from bootstrapping.

For response surface-based sensitivity analysis, you can experiment with different input distributions by modifying the input section of your psuadeRSME.in (or psuadeRSIE.in, etc.) file or, if you use 'rssobol1' (or similar commands), modifying your psData file to be loaded in command line mode. In the former case, if you use any non-uniform distributions, you need to change the sampling method to MC.

More sophisticated quantitative sensitivity analyses involving input correlations (governed by some inequality constraints, not joint PDFs) are available. For example, there are two ways to include input constraints into main effect analysis:

1. If you are using replicated Latin hypercube, say, in psuadeRSME.in, the steps are:

   (a) Generate a sample (for example, run PSUADE on psuadeRSME.in with analysis turned off;

   (b) Apply input constraints to filter out infeasible sample points in the sample file from the last step;

   (c) launch PSUADE and run 'me'.

2. Alternatively, you can also use numerical integration ('rssobol1b'). In this case, you will need to turn on the 'rs_constraint' line in your sample data file (psData). The syntax is:

   ```
   analyzer rs_constraint = constrSample indexFile Lbound Ubound
   ```

   where 'constrSample' is another PSUADE sample, 'indexFile' file contains a subset of input indices for constraining, and 'Lbound' and 'Ubound' are lower and upper bounds of the feasible region. For example, if you desire to impose constraint on input 2 and 3 such that $0 < X_2 + X_3 < 1$ (say, X_1 is not involved in the constraints), then 'constrSample' should contain a sample for the function $Y = X_2 + X_3$; 'indexFile' should contain 2 and 3; 'Lbound'= 0; and 'Ubound'= 1. If X_1 is part of the 'constrSample' (say, the sample contains the functional relationship $Y = X_1 + X_2 + X_3$) but it is not involved in the constraints ($0 < X_2 + X_3 < 1$), the 'indexFile' file should contain input 1, 2, and 3, but the line corresponding to input 1 should begin with a '0' followed by a default value (0 in this case to mimic the desired function), and 'Lbound' and 'Ubound' should be set the same as before.

4.6 Mixed Aleatory-Epistemic Uncertainty Analysis

When some of the inputs are epistemic parameters (that is, their actual prior probability distributions are not known, and their distributions are best described by a range, which is not the same as uniform distribution), it is not sufficient just to display the output probability distribution. Rather it should be an ensemble of probability distributions with each

corresponding to the distributions due to variations in the aleatory parameters at some fixed epistemic parameter values drawn from their respective ranges. PSUADE provides the functionality to perform this analysis.

Since this analysis is computationally intensive, it is currently implemented on response surfaces only. So the first step in this analysis is to create a response surface (say you have done it and the sample for constructing the response surface is in psData). Next, you need to prescribe the probability distributions for the aleatoric parameters by modifying the 'INPUT' section in psData (and keep the epistemic variables with ranges - lower and upper bounds - only). Then, launch PSUADE in command line mode, load psData, and run 'ae_ua'. You will be asked to select which input parameters should be considered as epistemic. At the completion of this analysis, PSUADE will output a file called 'matlabaeua.m' for viewing the ensemble of cumulative distributions.

Another similar analysis is the so-called 'second-order uncertainty analysis', which generates an ensemble of probability distributions as a result of uncertainties about the input distribution parameters. For example, let a certain parameter have a normal distribution with mean and standard deviation 1.2 and 0.5, respectively. Suppose there is an uncertainty about the mean; then this analysis 'so_ua' draws samples from this second level parameter uncertainties and generates distributions for each. At completion, this command will create a 'matlabsoua.m' file for visualizing the second-order uncertainties.

4.7 Numerical Optimization

Let the function for numerical optimization be the two-dimensional Rosenbrock function:

$$Y = 100(X_2 - X_1^2)^2 + (1 - X_1)^2, \quad X_i \in [-2, 2].$$

The PSUADE input file for numerical optimization can be constructed as follow (here bobyqa is a public domain software developed by Michael Powell):

```
PSUADE
INPUT
   dimension = 2
   variable  1 X1  =   -2.0   2.0
   variable  2 X2  =   -2.0   2.0
END
OUTPUT
   dimension = 1
   variable 1 Y1
END
METHOD
   sampling = FACT
   num_samples = 9
END
APPLICATION
```

```
      driver = ./simulator
      opt_driver = ./simulator
   END
   ANALYSIS
      optimization method = bobyqa
      optimization num_local_minima = 3
      optimization max_feval = 10000
      optimization tolerance = 1.0e-4
      optimization print_level = 2
   END
END
```

In setting up the optimization, the following decisions are to be made:

1. Is this a global optimization problem with many possible local minima? If so, should multi-start optimization be use?

2. What should the initial guess(es) be?

3. Which optimizer to use?

4. What is the termination criterion?

The **METHOD** section specifies how initial guesses are to be generated. For example, if one random initial guess is to be used, the sampling method should be 'MC' and the number of sample point ('num_samples') should be set to 1. If multi-start optimization is used (activated by setting **num_local_minima** to be larger than 1), one can use the sampling methods available in PSUADE (for example, factorial sampling as in the above example with 9 points). Alternatively, a user can provide his/her own initial guess(es) in the **PSUADE_IO** section.

In the example above, the analysis begins with the evaluation of the 9 factorial sample points. Subsequently, 3 (since **num_local_minima** = 3 sample points with the lowest output values are selected as the starting points for a multi-start optimization. Since it is a bound-constrained optimization, a suitable optimizer is 'bobyqa'. The termination criteria are prescribed via the maximum number of function evaluations (10000 in this case) or a termination tolerance ($1e - 4$). **driver** points to an executable called **simulator** (which is used to evaluate the initial guesses); and **opt_driver** points to an executable used in the actual optimization.

The above example is located in the Examples/OptRosenbrock directory. Simply compile the **simulator.c** file and then run **psuade psuadeBobyqa.in** to see optimization in action.

There are other advanced features in optimization such as avoiding repeated function evaluations (this is very useful for restart in the case when the function evaluation is expensive).

4.8 Bayesian Calibration

Let the function for numerical optimization be the function:

$$Y = F(X; a, b)$$

where X is some model design parameter (e.g. dimensions of the design) and Y is the model output (assume the output is scalar to simplify discussion); and a and b are parameters in the function that are not precisely known except that they fall between 0 and 1 (that is, they are uncertain parameters which are to be calibrated to match observation data). Suppose we also have a set of observation data $D = \{X_i \tilde{Y}_i\}_{i=1}^M$ that may help guide the search for the true values of a and b. Suppose further that we have decided to set a to some fixed value a^* and search only for the best b that fits the data set D. One way to find the best b is to perform a deterministic numerical optimization. If the experimental data $\tilde{Y}_i$'s are noisy, an alternative is to perform a statistical inference, the most popular of which is the one based on the Bayes rule.

To perform Bayesian inference, we need to have the following:

1. a set of calibration parameters (b in this example) and their distributions (priors, e.g. uniform between 0 and 1),

2. an observation data set (D in this example) together with the corresponding observation errors (let D^* be D appended with its standard deviations, i.e. $D^* = Normal(D, \Sigma)$), and

3. a sample (e.g. a Latin hypercube of size N) for generating a response surface ($\tilde{F}$) to approximate F (response surface is needed because a typical Bayesian inference requires many function evaluations).

A few other decisions need to be made:

1. whether discrepancy modeling is to be included in the inference (see below), and

2. whether response surface errors are to be included in the inference.

To include discrepancy modeling, an additional sample is to be created (internally in PSUADE) that describes the differences between the observation data and the corresponding function values (model outputs) at the experimental design points. Thus, this additional sample has size M with sample input and output pairs represented by $\{X_i e_i\}_{i=1}^M$ where

$$e_i = \tilde{Y}_i - \tilde{F}(X_i; a = a^*, b = b^*),$$

and b^* is some carefully selected value of b (the best choice is the posterior mean of b but since its posterior mean is not known, it can be set to its prior mean or mode). PSUADE's Bayesian calibration provides an option to create and store this discrepancy sample to a file for further examination (e.g. response surface analysis).

Let $M = 4$ be the number of observations at the design points $\{X_i\}_{i=1}^4$, and let the observation noise be 0.1 (standard deviation). We will specify these information to facilitate the construction of the likelihood function by creating a file, say 'mcmcFile', that contains

```
PSUADE_BEGIN
4 1 1 1
1 <X_1> <Y_1> 0.1
2 <X_2> <Y_2> 0.1
3 <X_3> <Y_3> 0.1
4 <X_4> <Y_4> 0.1
PSUADE_END
```

The first and last lines are markers recognized by PSUADE. The second lines specifies that there are four observations, one output, and 1 design paramter (a in this example), which is parameter 1 in the INPUT section. The next four lines each consists of the data set number (in order from 1 to 4 in this example), the design parameter value (X_i), and the observation value (Y_i) and its error (0.1 in this case).

To create a sample from the simulator (function F), we generate a Latin hypercube sample of, for example, size 100. The PSUADE input file (say, 'psuadeRS.in') to generate the Latin hypercube sample is:

```
PSUADE
INPUT
   dimension = 3
   variable   1 X   = 0.0  1.0
   variable   2 A   = 0.0  1.0
   variable   2 B   = 0.0  1.0
END
OUTPUT
   dimension = 1
   variable 1 Y
END
METHOD
   sampling = LH
   num_samples = 100
END
APPLICATION
   driver = ./simulator
END
ANALYSIS
   printlevel 1
END
END
```

Again, simply run the following command

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

and then move the result data file 'psuadeData' to, say, 'simdata'. After the preparation steps have been completed (make sure to validate your response surface), Bayesian inference can be launched by:

```
[Linux] psuade
*** ********************************************************** ***
*** Welcome to PSUADE (version 1.7.8)                         ***
*** ********************************************************** ***
PSUADE - A Problem Solving environment for
         Uncertainty Analysis and Design Exploration
psuade> load simdata
load complete : nSamples = 100
               nInputs  = 2
               nOutputs = 1
psuade> ana_expert (use expert mode to turn on discrepancy mode)
psuade> rsmcmc
.....
===> Enter the spec file for building likelihood function : mcmcFile
.....
Output 1
Enter you choice (for response surface type) ? 0
Output 2
Enter you choice (for response surface type) ? 0
Output 3
Enter you choice (for response surface type) ? 0
Output 4
Enter you choice (for response surface type) ? 0
.....
<say yes to discrepancy modeling, use default for other options)
MCMC BEGINS
 10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
.....
MCMC completed
**********************************************************************
MCMC: matlabmcmc2.m file (2-input analysis) is ready.
**********************************************************************
psuade> quit
[Linux]
```

If the inference is completed successfully, a Matlab (or Scilab) file ('matlabmcmc2.m') will contain both the prior and posterior distributions for visualization. Embedded in this Matlab file are some useful inference information. For example, the best negative log likelihood value will be displayed in the title of the posterior plots. Large departure of this likelihood value

may indicate a strong systematic bias and thus the need for discrepany modeling. Also, the red vertical line in each single-parameter posterior plot (the plots on the diagonal) indicates the parameter value at the maximum likelihood point. Note that the maximum likelihood point does not necesarry reside at the peak of the individual single-parameter probability distributions (for example, in the case of multi-modal posteriors) because these distributions have been marginalized. In addition, an optional output (this option is available when the 'ana_expert' mode is turned on) from the inference is a posterior sample written to the 'MCMCPostSample' file. This file stores a large sample drawn from the parameters' posterior distributions. At the end of this file the maximum likelihood solution is included together with more information about the details of the likelihood function for further diagnosis (for example, these details can help pinpoint the anomalous experiments.)

One may ask: if the likelihood function is derived from many experiments each of which has its own set of sample points, can PSUADE perform inference on it? The answer is yes and here is how to do it:

- Instead of loading a single sample ('simdata' above), first strip this sample of the PSUADE_IO section but keep the input and output sections (that is, the calibration parameters and ranges should be the same for all experiments, and the output section defines the outputs for each experiment).

- After loading the 'modified' sample file, run 'rsmcmc' as before. PSUADE will recognize that a sample has not been given and the inference program will ask, for each output, a sample file that will be used to create the response surface for that output.

- After all output sample files have been entered and response surface types selected, the rest of the 'rsmcmc' command will be the same.

Another flexibility offered in PSUADE is the construction of the likelihood function. If users prefer to prescribe their own likelihood function, they can compute the negative log likelihood value (scaled by 2.0) as a scalar model output (by assimilating the experimental data instead of providing providing the data to PSUADE), and enter zero mean and standard deviation of 1.0 when prompted. PSUADE will take the user-provided negative log likelihood and create the likelihood value by squaring its difference with the provided mean, dividing the result by the square of the provided standard deviation, multiplying it by -1/2 and finally exponentiating the result.

In a more complex scenario, suppose a in the following function

$$Y = F(X; a, b)$$

is not a fixed parameter to be set at some default value but an uncertain parameter with a posterior probability distribution from previous experiments and thus is not to be calibrated in the present experiments. Applying Bayesian inference in this case requires marginalization or integrating out the uncertainties of a. PSUADE currently does not have the inherent capability to handle this case. However, this can still be achieved by superscripting - that is, generating a sample for a, running a superscript (e.g. in Python) that performs MCMC

multiple times each of which a different value of a is to be fetched from the sample, and aggregating the posterior samples from the multiple MCMC runs to form the final posterior sample.

There are other advanced features in PSUADE's MCMC method such as tuning MCMC parameters (e.g. number of chains, termination criterion) and support for inferences in the presence of all four types of parameters: design, calibration, fixed, and uncertain (this last type is supported via superscripting. Please consult PSUADE developers for further details).

In the default mode, PSUADE's 'rsmcmc' runs a brute-force statistical inference (that is, run a large sample drawn from the priors to generate the posteriors). This default option is in generally faster and gives more precise posteriors. Users, however, may choose the traditional Gibbs MCMC algorithm by issuing the command 'setconfigoption' (with option 'MCMC_gibbs') before running 'rsmcmc'. When the Gibbs method is selcted, inference can also be accelerated by running the multiple-chain MCMC in parallel. This capability will require PSUADE be compiled using MPI, an option called 'PARALLEL_BUILD' that can be turned on during installation.

4.9   Optimization Under Uncertainty (OUU)

Let the simulation model be represented by

$$Y = F(X, U, \omega, \theta)$$

which is characterized by four types of variables:

1. Design/Decision variables X are the optimization variables that will be tuned to optimize some objective function,

2. Recourse/Operational variables U are scenario variables which can be tuned in a given system operating under different conditions (different values of ω and θ),

3. Discrete/Scenario variables ω have an enumerable set of states (called scenarios) such that each state is associated with a probability (and sum of probabilities for all possible states is equal to 1), and

4. Continuous uncertain variables θ are associated with a joint probability density function.

In the context of the formulation above, two different types of OUU are possible:

1. Single-stage OUU: this formulation requires no recourse variables so that it is solving the following problem:
$$\min_X \Phi_{\omega,\theta}(F(X, \omega, \theta))$$

where $\Phi_{\omega,\theta}$, the objective function, is some statistical quantity (e.g. mean) from running $F(X, \omega, \theta)$ at some realizations (a sample) of ω and θ (either ω or θ can be an empty set but not both).

2. Two-stage OUU: this formulation requires recourse variables for inner optimization but the recourse variables may be defined implicitly (not needing to be declared). This type of OUU solves the following problem:

$$\min_{X}[\Phi_{\omega,\theta}(\min_{U} F(X, U, \omega, \theta))].$$

In essence, the first type is analogous to typical numerical optimization except that, in the presence of uncertainties, the objective function to be optimized is some average of the deterministic objective function. The second type, on the other hand, has the goal to obtain an optimized design (with respect to X) but this design can dynamically adapt to different operational conditions (U) to optimize its performance. Thus, the 2-stage method involves an inner-outer optimization loop. Users can optionally provide the function F in 'opt_driver' for which PSUADE will wrap around it with two layers of the BOBYQA optimizer (to optimize with respect to X in the outer loop and U in the inner loop), or provide the inner optimization ($\min_{U} F(X, U, \omega, \theta)$) for which PSUADE will just wrap around it with one layer of the BOBYQA optimer (to optimize with respect to X).

An example is given in Examples/OUU/Problem2. The PSUADE input file is 'ouu_opt_driver.in':

```
PSUADE
INPUT
   dimension = 12
   variable   1 D1  = -5      5.0
   variable   2 D2  = -5      5.0
   variable   3 D3  = -5      5.0
   variable   4 D4  = -5      5.0
   variable   5 X1  = -10     10
   variable   6 X2  = -10     10
   variable   7 X3  = -10     10
   variable   8 X4  = -10     10
   variable   9 W1  = -5     5
   variable  10 W2  = -5     5
   variable  11 W3  = -5     5
   variable  12 W4  = -5     5
END
OUTPUT
   dimension = 1
   variable 1 Y
END
METHOD
   sampling = MC
   num_samples = 1
   randomize
   random_seed = 41491431
```

```
   END
   APPLICATION
      opt_driver = optdriver
   END
   ANALYSIS
      optimization method = ouu
      opt_expert
   END
END
```

In this example, the first 4 variables are design variables (type 1), the next 4 are recourse variables (type 2), and the last 4 are continuous uncertain parameters (type 4). The optimization driver should be created by compiling the 'optdriver.c' file. To run OUU, simply do:

> `[Linux] psuade ouu_optdriver.in`

and you will be prompted with a few questions asking for the number of each of the 4 types of variables $(4, 4, 0, 4)$, the type of each declared variables, the choice of objective function (enter '1' - the expected value), the choice of the sample source for the type 4 variables (PSUADE-generated), option to use response surface for estimating the expected value (enter 'n'), the sampling method used for type 4 (enter Latin hypercube), the sample size (enter 200), to choose 'your own inner optimizer', and 'n' to the rest of the questions. You will then see the OUU optimization in action giving the best design variable (X - variable $1 - 4$) settings at the end.

The newest additions of the OUU suite are:

- OUU with linear constraints (Powell's LINCOA)

- OUU with nonlinear inequality constraints (Powell's COBYLA)

- OUU with no bound constraints (Powell's NEWUOA)

- OUU with bound constraints and derivatives (LBFGS)

- OUU with both continuous and integer parameters (NOMAD)

- Different objective functions: output mean, linear combination of output mean and standard deviation, and value-at-risk (VaR).

4.10   A More Comprehensive Example

Suppose we are given a simulation model with 2 uncertain parameters X_1 and X_2 such that $Y = F(X_1, X_2)$ and with some given default values for X_1 and X_2. Suppose we do not know the uncertain range for X_1 and we arbitrarily impose its uncertain range to be $+/-20\%$ of its default value. For X_2, we also impose an initial range of $+/-20\%$, but we have another

experiment that will help refine its uncertainty range. Our overall objective is to quantify the uncertainty and parameter sensitivity of this model.

The steps to achieve the objective are (in Examples/CompositeTest):

1. Compile the available experimental data for refining the uncertainty distribution of X_2 (in file called 'expdata2').

2. Acquire the model ('simulator2.c') to apply Bayesian inference to refine X_2 and compile it (to become 'simulator2').

3. Put together a PSUADE input file ('psuade2.in') for Bayesian inference:

```
PSUADE
INPUT
   dimension = 1
   variable 1 X2  =  0.4   0.6
END
OUTPUT
   dimension = 1
   variable 1 Y
END
METHOD
   sampling = FACT
   num_samples = 10
   randomize
END
APPLICATION
   driver = ./simulator2
END
ANALYSIS
   printlevel 1
END
END
```

4. Run PSUADE with 'psuade2.in' and rename **psuadeData** to **psData2**.

5. Launch PSUADE, load **psData2** and apply 'rsmcmc' in command line mode to generate a posterior sample for $X2$ (turn on 'ana_expert' mode).

```
[Linux] psuade
*** ******************************************************** ***
*** Welcome to PSUADE (version 1.7.8)                       ***
*** ******************************************************** ***
PSUADE - A Problem Solving environment for
```

```
            Uncertainty Analysis and Design Exploration
      psuade> load psData2
      load complete : nSamples = 100
                     nInputs  = 2
                     nOutputs = 1
      psuade> ana_expert
      psuade> rsmcmc
      .....
      ===> Enter the spec file for building likelihood function : expdata2
      Say 'no' to the next question.
      Output 1
      Enter you choice (for response surface type) ? 2
      Enter 10000 and 20 to the next 2 questions.
      Enter 1 and 0 next to select input 1 and terminate.
      Say 'no' to discrepancy modeling.
      Say 'yes' to create posterior sample.
      Enter 100, 3 and 1.05 to the next 3 questions.
      .....
      MCMC: input   1 value at peak of likelihood = 5.600000e-01
      MCMC: input   1 mean     = 7.689363e-01
      MCMC: input   1 std dev = 1.300829e-01
      MCMC iterations completed
      MCMC: matlabmcmc2.m file has been created.
      **************************************************************************
      MCMC: check the MCMCPostSample file for a posterior sample.
      psuade> quit
      [Linux]
```

6. Convert the MCMC posterior sample to PSUADE data format by using the 'iread' in command line mode and the 'write' to a PSUADE file (say, 'sample2').

7. Generate a large sample for X_1 by running **psuade psuade1.in** (generate sample only and no simulation) and rename 'psuadeData' to 'sample1'.

8. Now we need to combine the two different distributions from two different inputs - one drawn from some standard distribution ('sample1') and the other from the Bayesian posterior distribution ('sample2'). Concatenation of the two 1-parameter samples 'sample1' and 'sample2' via 'rand_draw2' will give a 2-parameter sample to be propagated through the simulation model (or its response surface).

```
      [Linux] psuade
      *** ******************************************************** ***
      *** Welcome to PSUADE (version 1.7.8)                       ***
      *** ******************************************************** ***
```

```
      PSUADE - A Problem Solving environment for
              Uncertainty Analysis and Design Exploration
      psuade> rand_draw2
      Enter name of the first file : sample1
      Enter name of the second file : sample2
      Size of the sample to be drawn : (1-2000000) 100000
      Store random sample to : (filename) newSample
      psuade> quit
      [Linux]
```

9. Prepare the original 2-parameter simulation model ('simulator.c') by compiling it (to become 'simulator').

10. Propagate the 2-parameter sample through the simulator by setting the driver field in 'newSample' to be 'simulator' and running PSUADE on 'newSample' (Alternatively, if 'simulator' is expensive to run, replace it with a small sample and a response surface type.) After the runs have been completed, rename 'psuadeData' (e.g. to 'pdata').

11. Launch PSUADE to compute uncertainties and sensitivities (e.g. 'ua'). Turn on 'ana_expert' mode for Matlab graphics. You can also try 'me', 'tsi', and/or 'ca'.

```
      [Linux] psuade
      *** ******************************************************** ***
      *** Welcome to PSUADE (version 1.7.8)                       ***
      *** ******************************************************** ***
      PSUADE - A Problem Solving environment for
              Uncertainty Analysis and Design Exploration
      psuade> load pdata
      psuade> ua
      .....
      psuade> me
      .....
      psuade> quit
      [Linux]
```

12. If desired, compare the uncertainty distribution with the sample set without using experimental data (by running PSUADE on 'psuade.in') to assess how the use of experimental data affects the output uncertainty.

4.11   Adaptive Design of Experiment

Although the UQ discipline has been around for over two decades, many application scientists are still skeptical of its usefulness. They often question the value added by expending large

computational resources for UQ analysis. They generally agree that model predictions are plagued with uncertainties, but question how accurately quantifying these uncertainties can lend much help in their causes.

The UQ community has realized that one important use of UQ is to guide design the next experimental campaign. Physical experiments are expensive, and it is in the interest of application scientists to design experiments that maximize the information needed to better understand the physical phenomenon at hand and thus to enable a better modeling strategy. Adaptive (or sequential) design of experiment (DoE) is a class of techniques for intelligently proposing new experiments.

PSUADE currently has some capabilities for adaptive DoE. Specifically, the adaptive response surface methods employ statistical techniques to identify good sample points in the parameter space to run additional simulations in order to improve the quality of the response surface. Adaptive DoE can utilize the same statistical framework in search of the next batch of experiments.

The steps for adaptive DoE using PSUADE is as follow:

1. Create and run simulations on an initial sample drawn from the design space.

2. Start PSUADE in command line mode:

   - Load the simulated sample
   - Run the a_refine command (you will be asked how many additional sample points to generate)
   - Store the resident sample (use the 'write' command), in which the new sample points are appended at the end of the PSUADE_IO section.

The method PSUADE uses to create these new points is based on prediction uncertainty at different locations over the parameter space. The ones with the largest prediction errors are candidates for additional simulations. There are many other methods for selecting new design points, and some of these methods will be added to PSUADE in the future.

5   Summary

PSUADE is intended to be a general-purpose toolkit for uncertainty quantification. Many enhanced features have been incorporated based on our experiences with its practical application to complex multi-physics models; and not all of these features have been comprehensively described in this document. Users are encouraged to go through all examples included in the software releases and give us feedback and suggestions on improving the manuals and also the software itself.