UQpy - Uncertainty Quantification with Python

Dimitris G. Giovanis, M. D. Shields

Johns Hopkins University, USA

---

1. Installing UQpy

Prerequisites: You have at least one Python interpreter 3.6+ properly installed on your computer. In order to get the latest experimental version of UQpy the code can be installed from Github directly as follows:

```
$git clone https://github.com/SURGroup/UQpy.git
$cd UQpy/
$pip install -r requirements.txt.
$python setup.py install.
```

The last command might need **sudo** prefix, depending on your python setup.

2. Overview

UQpy (Uncertainty Quantification (UQ) using python) is a software toolbox containing a collection of modules written in Python that provide standardized solutions for many UQ problems that occur in physical model. Connection between UQpy and the user-defined computational model is made with text-based and bash shell script(s) provided by the user. Execution of UQpy results in realizations of the parametric space of interest using advanced techniques, as well as evaluations the corresponding model responses. UQpy is entirely code-agnostic and gives users a fully functional tool for performing UQ with nearly any computational analysis code. UQpy performs submission, execution, monitoring and post-process analysis, specifically tailored to the

analysis tool and the available platform and thus, it is amenable to performing adaptive UQ methods. `UQpy` is written in the Python 3 programming language.

2.1. Compiled version of `UQpy`

We need to address the Windows version

2.2. Interpreted version of `UQpy`

The interpreted version of `UQpy` requires a Python shell supporting Python 3.6+ as well as several common Python libraries as well. After downloading and installing `UQpy`, the following `UQpy`-specific files are required and must be co-located in the subdirectory lib/**UQpy**, which is in the same directory as UQpy_cmd.py:

- UQpyModules.py - Contains various functions.
- SampleMethods.py - Contains the available sampling methods used for exploring the parameter space.
- ReadInputFile.py - Reads the necessary UQ Parameter data file in case of running `UQpy` via command line, and converts it to python variables.
- PDFs.py - Contains the percent point functions of all the supported distributions; any new distribution can be added here.

3. Using UQpy- Required files

`UQpy` may be run using either an Integrated Development Environment (IDE) used in computer programming, specifically for the Python language or via the command line. The interpreted version of `UQpy`, has been tested to run in IDE PyCharm 2017.3.3.

In order to use `UQpy` for evaluating the response of any computational model for a number of parameter realizations, `UQpy` requires three **executable**[1] bash shell scripts:

[1] `$chmod +x` name1*.sh

- name1*.sh for linking the analysis software to `UQpy`
- name2*.sh for converting the the file containing the parameter values (text-based file) into appropriate input file for the analysis code
- name3*.sh converting the result of the software analysis into an appropriate (text-based) file to be read from `UQpy`. This is necessary in case of running adaptive UQ methods and/or post-processing of the results.

The names of these files are user defined. Additional to these files, if the user wants to generate the realizations of the random parameters according to one of the available sampling methods provided in `UQpy`, it is necessary to provide an text-based file under the name (UQpy_params.txt), which will enclose all the probabilistic information required for running the selected sampling method. T

The aforementioned files are directly specified by the user and may be in any directory.

4. UQpy Usage

`UQpy` is user friendly since it only requires the user to have basic knowledge in writing bash shell scripts.

4.1. Using the `UQpy` Command Line Mode

`UQpy` can be executed directly through the command line. It is provided as an option to the user who doesn't have sufficient familiarity and experience with python. Command line execution is advantageous when analyses need to be performed on a high-performance computing systems without direct graphics capability. In order to execute the interpreted version `UQpy` from the command line the user needs to change to the `UQpy` directory and then type in terminal :

```
$python UQpy_cmd.py --dir pathToModel --model name1*.sh --input name2*.sh --output name3*.sh
```

Where

- UQpy_cmd.py is the python script that actually runs `UQpy` via command line and needs to be located in the directory **UQpy**.

- `--dir` is the absolute path to the folder which contains the necessary files {name1*.sh , name2*.sh , name3*.sh and UQpy_params.txt}.
- `--model` points to the name1*.sh bash script
- `--input` points to the name2*.sh bash script
- `--output` points to the name3*.sh bash script

In order for `UQpy` to run from command line the file UQpy_params.txt is necessary to be located inside `--dir` otherwise, the execution will return an error. However the user may skip the entries {`--input, --output, --model`} if `UQpy` is utilized only for generating realizations of the random parameter and not for model evaluations. Another optional entry for the user is `--CPUs` which sets the number of processors used for the evaluation of the model, in case of parallel processing. The user can see all the available options (Fig. 1) by typing in terminal

```
$python UQpy_cmd.py --help
```

which results in:

```
python UQpy.py --{options}

optional arguments:
  -h, --help            show this help message and exit
  --dir MODEL_DIRECTORY
                        Specify the location of the model's directory.
  --input INPUT_SHELL_SCRIPT
                        Specify the name of the shell script *.sh used to
                        transform the output of UQpy (UQpyOut_*.txt file) into
                        the appropriate model input file
  --output OUTPUT_SHELL_SCRIPT
                        Specify the name of the shell script *.sh used to
                        transform the output of the model into the appropriate
                        UQpy input file (UQpyInp_*.txt)
  --model SOLVER        Specify the name of the shell script used for running
                        the model
  --CPUs CPUS           Number of local cores to be used for the analysis
```

Figure 1:

4.2. Using the UQpy IDE Mode

After installation, UQpy is build in the local Pythons standard library and thus, it runs from any Integrated Development Environment (PyCharm, Atom, Eclipse, e.t.c) which provides code analysis and debugging. In order to use UQpy libraries in a project the user needs to import the specific module to its workspace. This can be done by writing in a python script

```
from UQpy import *
```

which will load all modules of UQpy. If a specific class from the sample methods (e.g Monte Carlo simulation) is required then the user can selectively load it to the project by typing

```
from UQpy.SampleMethods import MCS
```

This functionality of UQpy enables the independent usage of its modules, which makes UQpy a powerful tool for UQ analysis and communication between python and various computational codes of different nature. In order to generate 100 realizations of two random parameters using MCS the user needs to type:

```
from UQpy.SampleMethods import MCS
x = MCS(dimension=2, pdf_type=['Uniform', 'Uniform'])
     pdf_params=[[0, 1], [0, 1] ], nsamples=100)
```

This will create the object x with is properties:

1. pdf_type: type of distribution for each parameter
2. pdf_params: distribution parameters
3. nsamples: number of samples to be generated
4. dimension: number of random parameters
5. samples: generated samples in the parameter space
6. samplesU01: generated samples in the Uniform space, $U[0, 1]^{dimension}$

5. UQpy workflow

6. Templates for the required Files

The interaction between UQpy and any external solver is made with text-based files which are simple to process and easy to work with in python.

6.1. Probabilistic Parameter File

The file that keeps the probabilistic properties of the parameters should always be under the name:

UQpy_params.txt

Creating UQpy_params.txt is simple and straightforward; Each property that is required for the selected sampling method, is defined in a line that starts with a hash-tag (#), followed by a key-word and/or key-phrase (case sensitive) describing the property[2]. The file ends with the key-word `#end`. Under that line, the specific attributes of the property are defined, according to the UQpy available options. Thus, different sampling methods require different parameter file .

6.1.1. Required properties for various sampling methods

The properties that need to be specified by the user inside the parameter file in order to run different sampling methods, for exploring the parameter space. A summary of these properties is given next:

[2]The order that the properties are declared in UQpy_params.txt is not important .

Monte Carlo simulation		
Property	**Mandatory**	**Optional**
#method	⋆	
#number of samples	⋆	
#number of parameters	⋆	
#distribution type	⋆	
#distribution parameters	⋆	
#names of parameters		⋆
#SROM		**True** or **False**

Latin hypercube simulation		
Property	**Mandatory**	**Optional**
#method	⋆	
#number of samples	⋆	
#number of parameters	⋆	
#distribution type	⋆	
#distribution parameters	⋆	
#names of parameters		⋆
#criterion		⋆
#distance		⋆
#metric		⋆
#SROM		**True** or **False**

Stratified sampling		
Property	**Mandatory**	**Optional**
#method	⋆	
#distribution type	⋆	
#number of parameters		⋆
#distribution parameters	⋆	
#design	⋆	
#names of parameters		⋆
#SROM		**True** or **False**

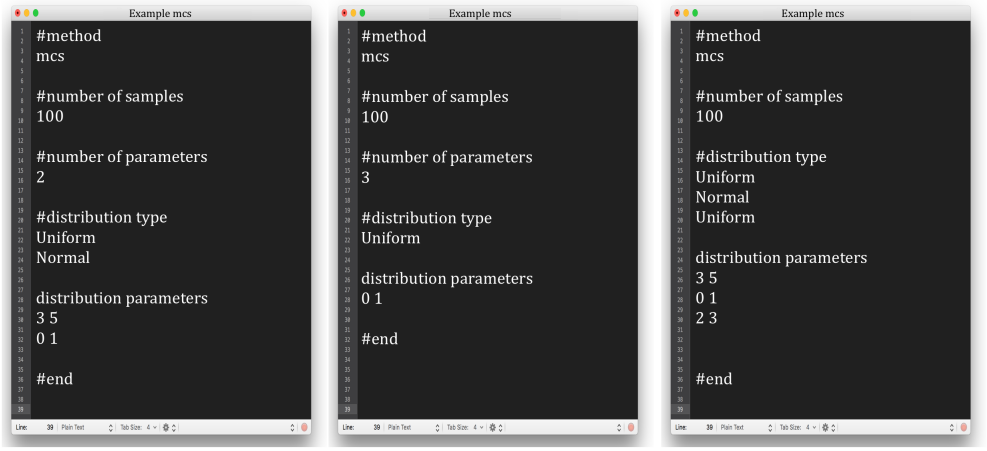
Partially Stratified sampling		
Property	**Mandatory**	**Optional**
`#method`	⋆	
`#distribution type`	⋆	
`#distribution parameters`	⋆	
`#number of parameters`		⋆
`#design`	⋆	
`#strata`	⋆	
`#names of parameters`		⋆
`#SROM`		**True** or **False**

Stochastic reduced order model		
Property	**Mandatory**	**Optional**
If `#SROM` property is **True**		
`#moments`	⋆	
`#error function weights`	⋆	
`#properties to match`		⋆
`#sample weights`		⋆

6.1.2. Examples of parameter files

Special instruction on how to create the parameter file that will enclose the required properties of the selected sampling method are the following :

- A complete parameter file for e.g. Monte Carlo simulation can defined like Fig.2(a).
- If all random parameters follow the same distribution type with the same distribution parameters then a parameter file can defined like Fig.2(b) where the distribution type and parameters need to be defined once. In this case existence of the property "number of random parameters" is mandatory.
- For the case the number of distribution type is equal to the number of distribution parameters (Fig.2(c)) then, definition of property "number of parameters" is optional .



(a) (b) (c)

Figure 2:

6.2. Template Input File

The functionality of the name2*.sh bash shell script file is to convert the text-based output file of `UQpy` (UQpy_run_i.txt) that contains the realization i of the parameter vector into appropriate input for the analysis code. The user is responsible for creating the appropriate bash script for performing this action. For example, if the software code reads a text-based file called modelInput_i.txt then a possible name2*.sh script would be the one depicted in Fig.3; it is used for renaming UQpy_run_i.txt to modelInput_i.txt.

6.3. Template Model File

In order for `UQpy` to execute the software code a bash script (name1*.sh) is necessary.

6.4. Template Output File

The functionality of the name3*.sh bash shell script file is to convert the output of the code analysis (which can be at any format) into a text file file under the name UQpy_eval_i.txt", where i refers to the number of simulation, ready to be processed by `UQpy`. This step is required for running adaptive

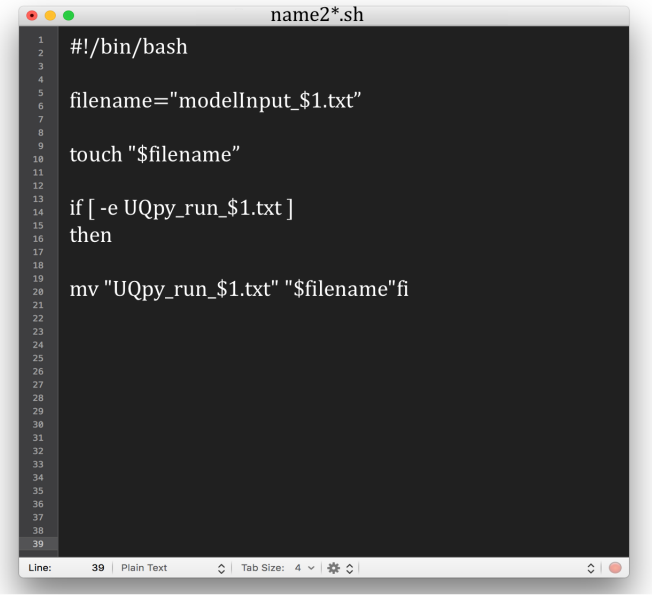


Figure 3:

UQ methods as well as for post-processing of the result but in any case it is mandatory to provide such file. For example, if the software code generates a text-based file called solution_i.txt then a possible name3*.sh script would be the one depicted in Fig.5; it is used for renaming solution_i.txt to UQpy_eval_i.txt".

7. `UQpy` Modules, Classes, & Functions

`UQpy` is structured in five core modules, each centered around specific functionalities:

1. `SampleMethods`: This module contains a set of classes and functions to draw samples from random variables. These samples may be randomly drawn, as in Monte Carlo simulation, or they may be deterministically drawn as in stochastic collocation or quasi-Monte Carlo.
2. `Inference`: This module contains a set of classes and functions to conduct probabilistic inference. The module contains methods that are based on Bayesian, frequentist, likelihood, and information theories.

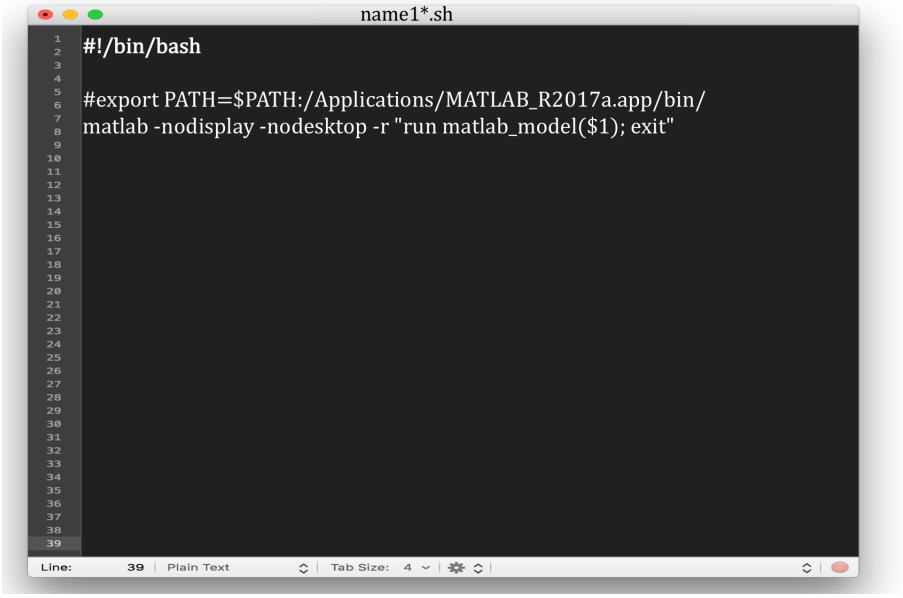


Figure 4:

3. `Reliability`: This module contains a set of classes and functions designed specifically to estimate probability of failure.
4. `Surrogate`: This module contains a set of classes and functions for building surrogate models, meta-models, or emulators.
5. `RunModel`: This module contains a set of classes and functions that allows `UQpy` to initiate simulations using either python or third-party computational solvers.

The following sections detail the classes and functions in each module with reference to examples that illustrate their use. Guidance is based on usage in IDE Model (see Section 4.2)

7.1. `SampleMethods` Module

The `SampleMethods` module consists of classes and functions to draw samples from random variables. It is imported in a python script using the following command:

```
from UQpy import SampleMethods
```

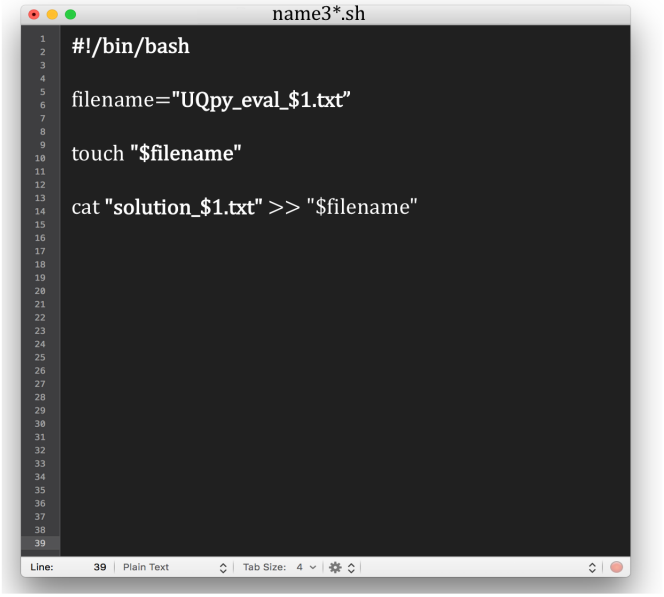


Figure 5:

The `SampleMethods` module has the following classes, each corresponding to a different sampling method:

Class	Description
`MCS`	Monte Carlo Sampling
`LHS`	Latin Hypercube Sampling
`STS`	Stratified Sampling
`PSS`	Partially Stratified Sampling
`MCMC`	Markov Chain Monte Carlo
`SROM`	Stochastic Reduced Order Model

Each class can be imported individually into a python script. For example, the MCMC class can be imported to a script using the following command:

```
from UQpy.SampleMethods import MCMC
```

The following subsections describe each class, their respective inputs and attributes, and their use.

7.1.1. `UQpy.SampleMethods.MCS`

7.1.2. `UQpy.SampleMethods.LHS`

Property	**Type**	**Options**
`#criterion`	*string*	'random', 'centered', 'maximin', 'correlate'
`#distance`	*string*	'braycurtis', 'canberra', 'chebyshev', 'cosine', 'dice', 'euclidean', 'hamming', 'jaccard', 'cityblock', 'matching', 'minkowski', 'rogerstanimoto', 'correlation', 'sokalmichener', 'sokalsneath', 'sqeuclidean', ' 'kulsinski', 'mahalanobis', 'russellrao', 'seuclidean',

7.1.3. `UQpy.SampleMethods.STS`

7.1.4. `UQpy.SampleMethods.PSS`

7.1.5. `UQpy.SampleMethods.MCMC`

The attributes of the `MCMC` class are listed below:

`MCMC` Class Attributes			
Attribute	**Input/Output**	**Required**	**Optional**
`dimension`	Input		$\star$
`pdf_proposal_type`	Input		$\star$
`pdf_proposal_scale`	Input		$\star$
`pdf_target_type`	Input		$\star$
`pdf_target`	Input	$\star$	
`pdf_target_params`	Input		$\star$
`algorithm`	Input		$\star$
`jump`	Input		$\star$
`nsamples`	Input	$\star$	
`seed`	Input		$\star$
`nburn`	Input		$\star$
`samples`	Output		

A brief description of each attribute can be found in the table below:

MCMC Class Attributes			
Attribute	**Type**	**Options**	**Default**
`dimension`	*integer*		`dimension` = 1
`pdf_proposal_type`	*string*	'Normal' 'Uniform'	'Uniform'
`pdf_proposal_scale`	*float* *float list*		[1,1,...,1] len = `dimension`
`pdf_target_type`	*string*	'marginal_pdf' 'joint_pdf'	'marginal_pdf'
`pdf_target`	*function* *string*		
`pdf_target_params`	Input		$\star$
`algorithm`	Input		$\star$
`jump`	Input		$\star$
`nsamples`	Input	$\star$	
`seed`	Input		$\star$
`nburn`	Input		$\star$
`samples`	Output		

Property	**Type**	**Description/Options**
`#criterion`	*string*	'random', 'centered', 'maximin', 'correlate'
`#distance`	*string*	'braycurtis', 'canberra', 'chebyshev', 'cosine', 'dice', 'euclidean', 'hamming', 'jaccard', 'cityblock', 'matching', 'minkowski', 'rogerstanimoto', 'correlation', 'sokalmichener', 'sokalsneath', 'sqeuclidean', ' 'kulsinski', 'mahalanobis', 'russellrao', 'seuclidean',

Property	**Options**
`#target distribution`	'multivariate_pdf', 'marginal_pdf', 'normal_pdf'
`#proposal distribution`	'Uniform', 'Normal'
`#algorithm`	'MH', 'MMH'

7.1.6. `UQpy.SampleMethods.SROM`

7.1.7. Adding a sampling method in UQpy

References