UQLab: the uncertainty quantification software framework

Bruno Sudret

Weimar, September 5th, 2016
Computational models in engineering

Complex engineering systems are designed and assessed using computational models, a.k.a simulators.

A computational model combines:

- A mathematical description of the physical phenomena (governing equations), e.g. mechanics, electromagnetism, fluid dynamics, etc.

- Discretization techniques which transform continuous equations into linear algebra problems.

- Algorithms to solve the discretized equations.

\[
\nabla \cdot \mathbf{D} = \rho \\
\nabla \cdot \mathbf{B} = 0 \\
\n\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \\
\n\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t}
\]
Real world is uncertain

- Differences between the \textit{designed} and the \textit{real} system:
  - Dimensions (tolerances in manufacturing)
  - Material properties (\textit{e.g.} variability of the stiffness or resistance)

- \textbf{Unforecast exposures:} exceptional service loads, natural hazards (earthquakes, floods, landslides), climate loads (hurricanes, snow storms, etc.), accidental human actions (explosions, fire, etc.)
What is uncertainty quantification?

- Input Variables \( X \in \mathcal{D}_X \)
- Computational Model \( M \)
- Quantities of Interest \( Y = M(X) \)

- What is the **scattering** of a quantity of interest \( Y \)?
- What are the parameters that drive the uncertainty on the QoI?
- What is the **probability of failure** (resp. non performance) of the system?
- What is the **optimal design** (e.g. minimal cost) that guarantees some performance?
- What are the **best-fit** model parameters that allow one to reproduce experimental data?

PDF \( f_Y \)

\[ \hat{\mu}_Y, \hat{\sigma}_Y \]

Sensitivity indices

\[ p_f = P(Y \geq y_{adm}) \]

\( d^* = \arg \min_d c(d) \quad \text{s.t.} \quad P(g(X(d), Z) \leq 0) \leq p_{f,adm} \)

Bayesian inversion
Outline

1. Introduction
2. Background of UQLab
3. Architecture and features
4. Demo
Why a new UQ software?

- Huge interest in UQ in the last few years
- (Few) well-established software
  - Covering only a limited spectrum of UQ techniques, e.g. sensitivity analysis, Kriging, etc.
  - Difficult to handle as a beginner/non-expert in the field, e.g. need for compiling a full platform (+ dependencies)
  - Too complex to use for engineers, e.g. built-in scripting language, C/C++ skills needed)
- Poor interactions with other existing and/or commercial software
- Non-modular structure requiring highly intrusive development to extend functionality (if possible at all)
- Limited portability
- Lack of / limited HPC capabilities
- Poor/complex documentation
Based on previous experience, the following requirements have been set:

- General purpose UQ software composed of modules that can transparently interact
- Set up / configuration time limited to minutes
- User interface through close-to-natural language (for end users). Learning curve steep (first analysis within one hour)
- Open source scientific algorithms
- Identical on windows/linux/mac systems
- Thought for high performance computing
- Comprehensive and accurate documentation
Programming language

Requirements

- No compiled language
- High-level pre-existing objects and built-in functions
- Mature environment for easy install
- Well-known in academia AND in the industry

Advantages

- Quick learning curve
- Standard in academia and industry, well supported
- Object-oriented
- Fast: based on LAPACK libraries
- Portable (Win, MacOS, Linux)
- Powerful debugging-profiling

Drawbacks

- Paying licenses for Matlab (inexpensive for academics)
- HPC support may require additional licenses
- High level libraries not part of basic Matlab
History of development

2011-12  First thoughts on a generic platform suitable for research and dissemination to industry

Nov. 2012  Kick-off meeting

Jan.-June 2013  Development of the core (Dr. Stefano Marelli)

2013-2015  Development, documentation and validation of the main modules

INPUT (incl. sampling) / PCE / Kriging / Sensitivity analysis

July 1st, 2015  Release of the beta version V0.9

Mar. 1st, 2016  Release of the Structural Reliability module (V0.92)

Oct. 2016  Release of open-source version V1.0
Facts and figures

http://www.uqlab.com

- Release of V0.9 on July 1st, 2015
  ETH license, free of charge for academia

- **450 active licences in 46 countries**

- About 60% license renewal after one year

<table>
<thead>
<tr>
<th>Country</th>
<th># licences</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>83</td>
</tr>
<tr>
<td>France</td>
<td>66</td>
</tr>
<tr>
<td>Switzerland</td>
<td>55</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>24</td>
</tr>
<tr>
<td>China</td>
<td>22</td>
</tr>
<tr>
<td>Italy</td>
<td>21</td>
</tr>
<tr>
<td>India</td>
<td>19</td>
</tr>
<tr>
<td>Belgium</td>
<td>16</td>
</tr>
<tr>
<td>Germany</td>
<td>14</td>
</tr>
<tr>
<td>Brazil</td>
<td>12</td>
</tr>
</tbody>
</table>
UQLab users
UQLab
The Framework for Uncertainty Quantification

"Make uncertainty quantification available for anybody, in any field of applied science and engineering"

- MATLAB-based Uncertainty Quantification framework
- State-of-the-art, highly optimized algorithms
- Easy to use and deploy
- Designed to be extended by users
Outline

1. Introduction

2. Background of UQLab

3. Architecture and features
   - Generic rules
   - **Model** module
   - **Input** module
   - Surrogate modelling
   - **Sensitivity** and **Reliability** modules

4. Demo
An uncertainty quantification problem is defined by:

- **A computational model** of the physical system
- **A probabilistic model** of the input uncertainties
- **Analysis algorithms**
- **The Gateway** is the entry point/memory management unit
- **Core modules** (INPUT/ MODEL/ ANALYSIS) keep trace of the objects created by the users (bookkeeping)
- Each object efficiently stores information needed by the underlying algorithms
UQLab: an implementation of the global UQ framework

<table>
<thead>
<tr>
<th>Mathematics</th>
<th>UQLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational model</td>
<td>$y = M(x)$</td>
</tr>
<tr>
<td>Probabilistic description of input uncertainty</td>
<td>$X \sim f_X$</td>
</tr>
<tr>
<td>Monte Carlo simulation</td>
<td>$\hat{\mu}_Y, \hat{\sigma}_Y^2, \hat{f}_Y$</td>
</tr>
<tr>
<td>Polynomial chaos expansions</td>
<td>$\sum_{\alpha \in A} y_{\alpha} \Psi_{\alpha}(X)$</td>
</tr>
<tr>
<td>Kriging</td>
<td>$\beta^T \cdot f(x) + \sigma^2 Z(x, \omega)$</td>
</tr>
<tr>
<td>Sensitivity analysis</td>
<td>Importance factors</td>
</tr>
<tr>
<td>Reliability analysis</td>
<td>$P_f = \mathbb{P}(M(X) \leq \bar{m})$</td>
</tr>
</tbody>
</table>
Defining a UQLab problem

- Launch Matlab, start the software by typing `uqlab`
- Create Input and Model objects by
  - defining their properties
  - using the `uq_createXXX` command

```matlab
% Model
M0pts.mFile = 'myComputationalModel';
myModel = uq_createModel(M0pts); % create MODEL object

% Input random vector
RV.Marginals(1).Type = 'Uniform';
RV.Marginals(1).Parameters = [-pi, pi];
RV.Marginals(2).Type = 'Lognormal';
RV.Marginals(2).Moments = [5, 0.8];
...
myInput = uq_createInput(RV); % create INPUT object

% Define analysis options and run it

An0pts.Type = 'Sensitivity';
An0pts.Method = 'Sobol';
ResObject = uq_createAnalysis(An0pts); % compute Sobol' indices
```
Storing data in objects

- The commands `uq_createXXX` create objects which store information in a structured way, eg:

```plaintext
myInput = uq_input with properties:
    Type: 'uq_default_input'
    Name: 'Input 1'
    Internal: [1x1 struct]
    Marginals: [1x3 struct]
    Copula: [1x1 struct]
    Sampling: [1x1 struct]
    Options: [1x1 struct]

myModel = uq_model with properties:
    Name: 'Model 1'
    isVectorized: 1
    Parameters: []
    mFile: 'myComputationalModel'
```
Reading / plotting results

- When running an Analysis, the results are stored in a Matlab variable e.g. ResObject

- Fields of the Analysis outcome can also be read directly:

  ```matlab
  >> ResObject.Results
  Total: [8x1 double]
  FirstOrder: [8x1 double]
  TotalVariance: 832.7455
  Cost: 100000
  ExpDesign: [1x1 struct]
  PCEBased: 0
  VariableNames: {'rw' 'r' 'Tu' 'Hu' 'Tl' 'Hl' 'L' 'Kw'}
  ```

- The main features can be printed on screen using `uq_print(ResObject)`

- Context-dependent plots can be obtained using `uq_display(ResObject)`
Most analyses require only that a Model and an Input object have been defined.

For parametric studies, different Model / Input / Analysis may however coexist in the workspace.

To avoid confusion, the ones used are stored in an Internal field of the ResObject.

```
>> ResObject.Internal
    Type: 'Sensitivity'
    Method: 'sobol'
    Sobol: [1x1 struct]
    Model: [1x1 uq_model]
    Input: [1x1 uq_input]
```

NB: In case several ones have been defined, the last defined is the currently used.
Bookkeeping of Inputs and Models

- A list of existing Input, Model and Analysis objects in the workspace can be obtained using uq_listInputs, uq_listModels, uq_listAnalyses

- The active one is tagged with a “>” sign

```plaintext
>> uq_listInputs
Available input objects:
  1) Input 1
  > 2) Correlated input
```

- The current active Input/Model/Analysis may be changed by using uq_selectInput/uq_selectModel/uq_selectAnalysis

```plaintext
>> uq_selectModel
Available model objects:
  1) R-S model
  > 2) R-S model (vectorized)
Please select a model:
```
Outline

1 Introduction

2 Background of UQLab

3 Architecture and features
   Generic rules
   **Model** module
   **Input** module
   Surrogate modelling
   **Sensitivity** and **Reliability** modules

4 Demo
Computational models

Analytical functions

```matlab
% String
MOpts.mString = 'X(1) - X(2)'

% Matlab function handle
MOpts.mHandle = @(X) X(1) - X(2) % non vectorized
MOpts.mHandle = @(X) X(:,1) - X(:,2) % vectorized

% Matlab m-file
MOpts.mFile = 'myComputationalModel'
```

where:

```matlab
function Y = myComputationalModel(X)
Y = X(:,1) - X(:,2) % vectorized
```
Computational models

Matlab native code: packaged as a m-file

```matlab
function Y = myQuantityOfInterest(X)
...
...  % some complex code
Y = ...;
```

Wrapper of an external code

```matlab
function Y = myQuantityOfInterest(X)
  % pre-process the current input values
  PreProcess(X);
  % write the input file to the third party code
  WriteInputFile;
  % run it from within Matlab
  !ThirdPartyCode.exe theInputFile > theOutputFile
  % read quantities of interest from the output file
  Y = PostProcess('TheOutputFile');
```

No model: pre-computed data sets

- Surrogate models are built up from experimental designs that may be computed outside UQLab
- Several recent projects have been carried out without any physical link between UQLab and third party codes
Computational models

Precomputed surrogate model

- Polynomial chaos expansion
- Kriging
- PC-Kriging
- Support vector machines
- Low-rank tensor approximations
1 Introduction

2 Background of UQLab

3 Architecture and features
   Generic rules
   MODEL module
   INPUT module
   Surrogate modelling
   SENSITIVITY and RELIABILITY modules

4 Demo
Probabilistic models: **INPUT module**

The uncertain input variables are modelled by a random vector $X$ defined by:

- a set of **marginal distributions**:
  - 10 parametric distributions incl. Gaussian, lognormal, Gamma, Weibull, etc.
  - Non parametric data-based distributions from **kernel smoothing**
  - User-defined (supplied by PDF, CDF and inverse CDF functions)

- a **copula function** for modelling dependence (Gaussian, elliptic)

*Reminder: according to Sklar’s theorem, the joint cumulative distribution function may be defined through its marginals and copula*

$$F_X(x) = C \left( F_{X_1}(x_1), \ldots, F_{X_M}(x_M) \right)$$

where:

- $F_X(x)$ is the joint CDF
- $F_{X_i}$’s are marginal distributions
- $C$ is the copula function

```matlab
% Input random vector
RV.Marginals(1).Type = 'Uniform';
RV.Marginals(1).Parameters = [-pi, pi];
RV.Marginals(2).Type = 'Lognormal';
RV.Marginals(2).Moments = [5, 0.8];
...
myInput = uq_createInput(RV); % create input vector`
Sampling random vectors

Once an INPUT object is defined, it can be sampled:

\[
\begin{align*}
X_1 &= \text{uq\_getSample}(100); & \quad \% \text{ 100 realizations of current INPUT} \\
X_2 &= \text{uq\_getSample}(\text{myInput}, 100); & \quad \% \text{ also specifying the INPUT object}
\end{align*}
\]

Sampling methods in the unit hypercube

- Crude Monte Carlo
- Latin hypercube sampling (+ maximin)
- Quasi-random sequences: Sobol’ and Halton points

\[
X_2 = \text{uq\_getSample}(200, 'Sobol') \quad \% \text{ 200 Sobol’ points from current INPUT}
\]

Isoprobabilistic transform

- Generalized Nataf transform

Enrichment

- \text{enrichLHS}, \text{enrichSobol}, \text{enrichHalton}, \text{lhsify} features
Outline

1. Introduction

2. Background of UQLab

3. Architecture and features
   - Generic rules
   - MODEL module
   - INPUT module
   - Surrogate modelling
   - SENSITIVITY and RELIABILITY modules

4. Demo
Polynomial chaos expansions

The PCE module allows to build up and use PCE of the form:

\[ \tilde{M}(x) = \sum_{\alpha \in A} y_{\alpha} \Psi_{\alpha}(x) \]

Generalized PCE basis

- Classical orthogonal polynomials (Hermite, Legendre, Laguerre, Jacobi)
- Arbitrary polynomials
- Built-in isoprobabilistic transform \( X \rightarrow U \) (e.g. for lognormal inputs)

Truncation schemes

- Total degree
- Maximum rank
- Hyperbolic q-norm.
- Any combination
Polynomial chaos expansions

**Coefficients computation**

- Projection
  - Full quadrature
  - Smolyak sparse grids
- Ordinary least-squares (a.k.a. regression)
- Sparse PCE by compressive sensing:
  - Least-angle regression (LAR)
  - Orthogonal matching pursuit (OMP)

**Output**

- PCE (sparse) basis and coefficients
- Moments, LOO error
- Function handle to the PCE surrogate for reuse
Kriging (a.k.a Gaussian process modelling)

The Kriging module allows to build up and use Gaussian process emulators of the form:

\[
\tilde{M}(x) = \beta^T \cdot f(x) + \sigma^2 Z(x, \omega)
\]

**Trend**
- Ordinary Kriging (constant trend)
- Universal Kriging (linear, polynomial, PCE trend)
- User-defined (allows for hierarchical Kriging)

**Covariance kernels**
- Classical 1D kernels: exponential, square-exponential, Matérn + nugget option
- Separable and ellipsoidal multivariate kernels
- User-defined (also non stationary)
Kriging (a.k.a Gaussian process modelling)

Estimation methods
- Maximum likelihood
- Leave-one-out and leave-$K$-out cross validation

Optimization algorithms
- Gradient-based (BFGS)
- Genetic algorithm (Matlab ga)
- Hybrid (HGA)

Output
- Mean predictor and Kriging variance (optionally: covariance matrix)
- LOO error
Outline

1 Introduction

2 Background of UQLab

3 Architecture and features
   Generic rules
   MODEL module
   INPUT module
   Surrogate modelling
   SENSITIVITY and RELIABILITY modules

4 Demo
Sensitivity analysis

Linear(ized) methods

- I/O correlation and rank correlation coefficients
- Standard regression coefficients (SRC)
- Perturbation method (Taylor series expansion)
- Cotter method

Global sensitivity analysis

- Morris method
- Sobol’ indices (various MCS estimates)
- PCE-based Sobol’ indices
Reliability analysis (a.k.a. rare events estimation)

Approximation methods
- FORM
- SORM

Simulation methods
- Crude Monte Carlo
- Importance sampling
- Subset simulation

Surrogate-based methods
- AK-MCS (active learning based on universal Kriging)
- PCK-MCS (active learning based on PC-Kriging)
High Performance Computing

Objectives

- Provide an interface to common HPC resources
- **Local-like execution**: no need to manually connect and set-up scheduler scripts and retrieve the results
- Automated book-keeping
- **Non-intrusive** in user scripts

**UQLab dispatcher module**

- Simple text-based credentials
- Supports linux clusters (TORQUE)
- SSH-based implementation
- Non-intrusive
- “Local” feel

```matlab
% Create a dispatcher
DOpts.Profile = 'myCredentials';
DOpts.NCores = 2;
uq_createDispatcher(DOpts);

% Create a model
MOpts.mString = 'X(1)-X(2)';
myModel = uq_createModel(MOpts);

% Evaluate the model
X = [rand(100,1) rand(100,1)];
Y = uq_evalModel(X);```
Outline

1 Introduction

2 Background of UQLab

3 Architecture and features

4 Demo
   Simple Monte Carlo simulation
   Sensitivity analysis
   Polynomial chaos expansions
   Reliability analysis
   Kriging
**DEMO_01 – Simple Monte Carlo simulation**

**Problem statement**

Model: \( \mathbf{X} = (R, S)^T \); \( \mathcal{M}(\mathbf{X}) = R - S \)

Input: \( R \sim \mathcal{N}(5, 0.8) \); \( S \sim \mathcal{N}(2, 0.6) \)

Case 1: independent variables

Case 2: linear correlation \( \rho_{R,S} = 0.6 \)

**Questions:** mean / std.deviation of \( R - S \), PDF, \( \mathbb{P}(M \leq 0) \)

**Solution**

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>No correlation</td>
<td>( 5 - 2 = 3 )</td>
<td>( \sqrt{0.8^2 + 0.6^2} = 1 )</td>
</tr>
<tr>
<td>Correlation ( \rho_{R,S} = 0.6 )</td>
<td>( 5 - 2 = 3 )</td>
<td>( \sqrt{53/125} = 0.651152 )</td>
</tr>
<tr>
<td>( \mathbb{P}(M \leq 0) )</td>
<td>( \Phi(-3) \approx 1.35 \cdot 10^{-3} )</td>
<td>( 2.04 \cdot 10^{-6} )</td>
</tr>
</tbody>
</table>
DEMO_02 – Sensitivity analysis

Model: Borehole function

\[ M(x) = \frac{2\pi T_u (H_u - H_l)}{\ln(r/r_w) \left( 1 + \frac{2LT_u}{\ln(r/r_w)r_w^2 K_w} + \frac{T_u}{T_l} \right)} \]

\[ M(x) = \frac{2\pi T_u (H_u - H_l)}{\ln(r/r_w) \left( 1 + \frac{2LT_u}{\ln(r/r_w)r_w^2 K_w} + \frac{T_u}{T_l} \right)} \]

Input

<table>
<thead>
<tr>
<th>Variable</th>
<th>Distribution</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius of borehole (m) ( r_w )</td>
<td>Gaussian</td>
<td>( \mu = 0.1 ) ( \sigma = 0.0161812 )</td>
</tr>
<tr>
<td>Radius of influence (m) ( r )</td>
<td>Lognormal</td>
<td>( \lambda = 7.71 ) ( \zeta = 1.0056 )</td>
</tr>
<tr>
<td>Transmissivity of upper aquifer (m²/yr) ( T_u )</td>
<td>Uniform</td>
<td>( [63,070; 115,600] )</td>
</tr>
<tr>
<td>Potentiometric head of upper aquifer (m) ( H_u )</td>
<td>Uniform</td>
<td>( [990; 1,110] )</td>
</tr>
<tr>
<td>Transmissivity of lower aquifer (m²/yr) ( T_l )</td>
<td>Uniform</td>
<td>( [63.1; 116] )</td>
</tr>
<tr>
<td>Potentiometric head of lower aquifer (m) ( H_l )</td>
<td>Uniform</td>
<td>( [700; 820] )</td>
</tr>
<tr>
<td>Length of borehole (m) ( L )</td>
<td>Uniform</td>
<td>( [1,120; 1,680] )</td>
</tr>
<tr>
<td>Hydraulic conductivity of borehole (m/yr) ( K_w )</td>
<td>Uniform</td>
<td>( [9,855; 12,045] )</td>
</tr>
</tbody>
</table>

Questions: Sensitivity indices obtained by different methods
DEM0_03 – Polynomial chaos expansions

10 independent input variables
- 4 describing the bars properties
- 6 describing the loads

Questions

PDF of the max. deflection, statistical moments, probability of failure

\[ V = \mathcal{M}^{\text{FE}} (E_1, E_2, A_1, A_2, P_1, \ldots, P_6) \]

Probabilistic model

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Name</th>
<th>Distribution</th>
<th>Mean</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young’s modulus</td>
<td>( E_1, E_2 ) (Pa)</td>
<td>Lognormal</td>
<td>2.10 \times 10^{11}</td>
<td>2.10 \times 10^{10}</td>
</tr>
<tr>
<td>Hor. bars section</td>
<td>( A_1 ) (m^2)</td>
<td>Lognormal</td>
<td>2.0 \times 10^{-3}</td>
<td>2.0 \times 10^{-4}</td>
</tr>
<tr>
<td>Vert. bars section</td>
<td>( A_2 ) (m^2)</td>
<td>Lognormal</td>
<td>1.0 \times 10^{-3}</td>
<td>1.0 \times 10^{-4}</td>
</tr>
<tr>
<td>Loads</td>
<td>( P_1-P_6 ) (N)</td>
<td>Gumbel</td>
<td>5.0 \times 10^4</td>
<td>7.5 \times 10^3</td>
</tr>
</tbody>
</table>
Various PCE constructions

1. **Ordinary least-squares**: use of an experimental design of size twice the number of PCE coefficients
   - PCE of degree 2
   - PCE of degree 3
   - ...

2. **Sparse PCE with LARS**, fixed Sobol’ points experimental design of size 128, candidate bases of max. degree 2-6

3. **Sensitivity analysis** (PC-based Sobol’ indices)

4. **PCE** from existing data base
Limit state function: Hat function

\[ g(x) = 10 - (x_1 - x_2)^2 - 8(x_1 + x_2 - 3)^3 \]

Input:

\[ X_i \sim \mathcal{N}(0, 1), \quad i = 1, 2 \]

Failure probability

\[ P_f = \mathbb{P}(g(X) \leq 0) \]

Reliability methods

- Crude Monte Carlo simulation
- FORM
- Importance sampling
- Subset simulation
- AK-MCS

Computational model

\[ x \mapsto x \sin x \quad \text{for} \quad x \in [0, 15] \]

Experimental design

Six points selected in the range \([0, 15]\) using Monte Carlo simulation:

\[ \mathcal{X} = \{0.6042, 4.9958, 7.5107, 13.2154, 13.3407, 14.0439\} \]
Kriging predictor

```matlab
Trend.Type = 'ordinary';  % Ordinary Kriging
Covariance.Type = 'matern-5_2';  % Matérn 5/2
EstimMethod = 'ML';  % Maximum likelihood
Optim.Method = 'BFGS';  % BFGS algorithm
```

\[ M(x) = x \sin(x) \]

![Graph of \( M(x) = x \sin(x) \)]

---

B. Sudret (Chair of Risk, Safety & UQ)
The Kriging variance gives a local error estimator about the accuracy of the predictor (NB: the variance is equal to zero at points of the experimental design)

Large confidence intervals correspond to a large (epistemic) uncertainty on the prediction
Questions?

Chair of Risk, Safety & Uncertainty Quantification
www.rsuq.ethz.ch

The Uncertainty Quantification Laboratory
www.uqlab.com

Thank you very much for your attention!