UQLab: a framework for uncertainty quantification in MATLAB

Author(s):
Marelli, Stefano; Sudret, Bruno

Publication date:
2014

Permanent link:
https://doi.org/10.3929/ethz-a-010238238

Originally published in:
https://doi.org/10.1061/9780784413609.257

UQLAB: a framework for Uncertainty Quantification in MATLAB

Stefano Marelli[1] and Bruno Sudret[2]

[1,2]Chair of Risk, Safety and Uncertainty Quantification, Institute of Structural Engineering, ETH Zürich Stefano-Franscini-Platz 5, CH-8093 Zürich, Switzerland; Tel: +41 44 633 06 70; E-Mail: [1]marelli@ibk.baug.ethz.ch, [2]sudret@ibk.baug.ethz.ch

ABSTRACT

Uncertainty quantification is a rapidly growing field in computer simulation-based scientific applications. The UQLAB project aims at the development of a MATLAB-based software framework for uncertainty quantification. It is designed to encourage both academic researchers and field engineers to use and develop advanced and innovative algorithms for uncertainty quantification, possibly exploiting modern distributed computing facilities. Ease of use, extendibility and handling of non-intrusive stochastic methods are core elements of its development philosophy. The modular platform comprises a highly optimized core probabilistic modelling engine and a simple programming interface that provides unified access to heterogeneous high performance computing resources. Finally, it provides a content-management system that allows users to easily develop additional custom modules within the framework. In this contribution, we intend to demonstrate the features of the platform at its current development stage.

INTRODUCTION

Uncertainty quantification through computer simulation is an intrinsically interdisciplinary field that has seen a rapid growth in the last decades. Its techniques are rooted at the boundaries between computer simulation-based engineering, applied mathematics, statistics and probability theory. Broadly speaking, it aims at identifying sources of uncertainty in each component of the simulation of physical quantities and propagating this uncertainty into the model responses. Such a general formulation comprises a number of approaches including, amongst others, structural reliability, sensitivity analysis, reliability-based design optimization and Bayesian techniques for calibration and validation of computer models.

A number of computational tools are readily available to tackle the uncertainty quantification problem to different degrees. These include both free software, like FERUM (Bourinet 2009), OpenTURNS (Andrianov et al. 2007) and Dakota (Eldred et al. 2004), and commercial, like COSSAN and Nessus), alternatives. A review on structural reliability software is available, *e.g.*, in (Schuëller 2006).

To the best of our knowledge, however, none of the existent software is specifically designed to be extended by the engineering research community. The use of powerful but complex languages like C++ (Dakota) and Python (OpenTurns) as well as of Object Oriented programming paradigms often discourages relevant portions of the non-highly-IT trained scientific community from the adoption of otherwise powerful tools.

In an attempt to overcome such limitations, the Chair of Risk, Safety and Uncertainty Quantification in ETH Zürich has started the UQLAB (**U**ncertainty **Q**uantification in MAT**LAB**) project, with the objective of creating a powerful, modular and simple-to-extend software framework for uncertainty quantification (UQ) in engineering applications.

The main defining goals of the UQLAB platform are:

- to provide a complete set of tools for uncertainty quantification in engineering applications;
- to ensure ease of use for students, academic researchers and field engineers;
- to design a modular structure that is easy to extend by non highly-IT-trained scientists;
- to provide high interoperability with existing third party software in a non-intrusive, "black box"-type approach;
- to ease the deployment of uncertainty quantification algorithms on a variety of high-performance computing (HPC) platforms.

Due to its capillary distribution in engineering environments and simple learning curve, MATLAB was chosen as the ideal language for the toolbox.

A GLOBAL UNCERTAINTY QUANTIFICATION FRAMEWORK

In order to build an uncertainty quantification software with a broad enough scope, a correspondingly general theoretical framework is required. The theoretical backbone of the UQLAB software lies in the global uncertainty framework developed by (Sudret 2007; DeRoquigny 2008), sketched in Figure 1.

According to this framework, the solution of any uncertainty quantification problem can be decomposed in the following steps:

Step A: Define the physical model and the quantities of interest for the analysis, *e.g.* displacements at critical points of a civil structure. It is a deterministic representation of an arbitrarily complex physical model, *e.g.* a finite element model (FEM).

Step B: Identify and quantify the sources of uncertainty in the system that serve as input for Step A. They are represented by a set of random variables and their joint probability density function (PDF).

Step C: Propagate the uncertainty through the model (step A) from the input random variables (identified in Step B), *e.g.* structural reliability analysis.

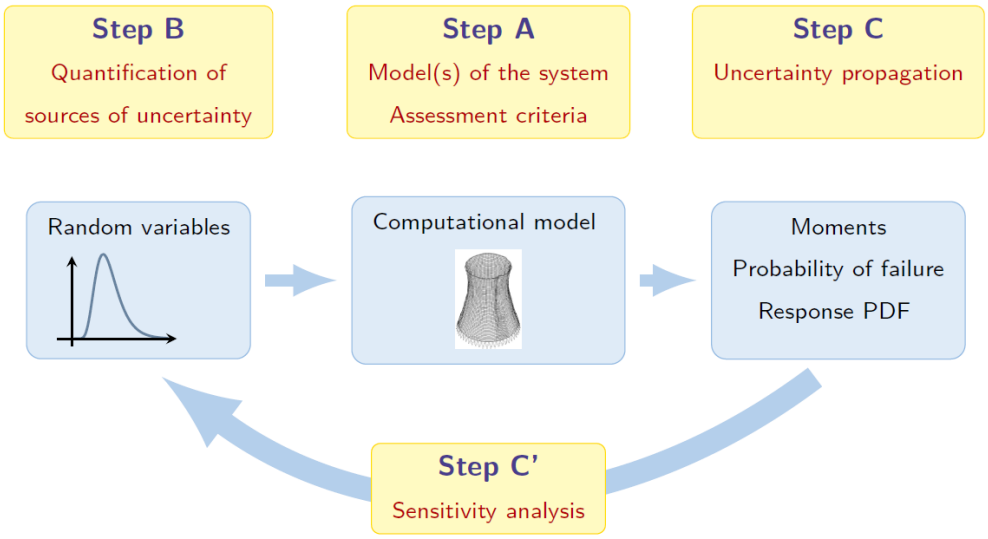


Figure 1. Visual representation of the global theoretical framework for uncertainty quantification at the basis of the UQLAB framework.

Step C': Optionally, exploit the by-products of the analysis in Step C to rank the sources of uncertainty according to their weight on the quantities of interest, *e.g.* sensitivity analysis.

These components introduce a clear semantic distinction between the actors involved in any UQ problem: a *computational model*, a description of the *random input parameters* (simply called "input") and various types of *uncertainty analysis*. This theoretical framework, therefore, provides the ideal foundation for the development of the information flow model in a multi-purpose uncertainty quantification software.

DESIGNING A SOFTWARE FRAMEWORK

The software architecture

To achieve in our software a level of generality and flexibility similar to that presented in the previous section, we decided to opt for the development of a computational framework, rather than a monolithic software package. A computational framework substantially differs from a "packaged software" in several important aspects:

- it focuses on the ability to create new features, in addition to providing them;
- its collaborative development model plays a relevant role in its architecture;
- its features must be arbitrarily extendible.

At the core of UQLAB lies a modular infrastructure that closely follows the semantics described in the previous section, graphically represented in Figure 2.

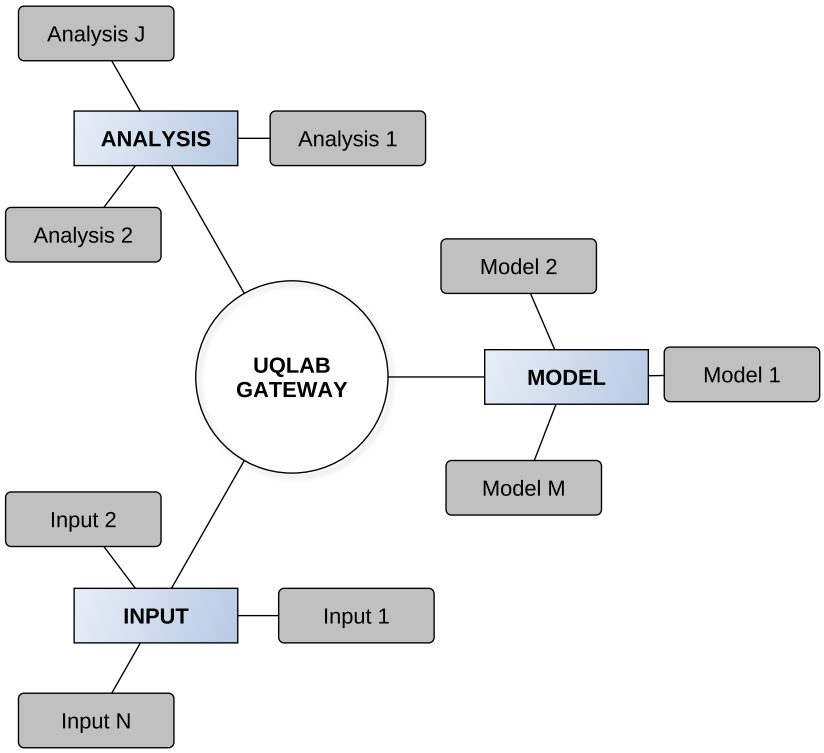


Figure 2. The modular core at the heart of the UQLAB framework architecture. An arbitrary number of elements can be connected at any stage of the uncertainty quantification problem.

The three steps identified in Figure 1 are directly mapped to *core modules* represented with light shaded boxes in Figure 2: **MODEL** corresponds to Step A (physical modelling), **INPUT** to Step B (sources of uncertainty) and **ANALYSIS** to Step C (uncertainty propagation). For the sake of simplicity, auxiliary *core modules* that handle additional software capabilities of the framework (*e.g.*, the dispatching of calculations to HPC resources), are not shown in Figure 2.

Each *core module* has several connections: one to a unique access point to the framework (the central **GATEWAY** in Figure 2), as well as an arbitrary number to leaf *modules* (grey boxes in Figure 2). Such connections are at the core of the information flow within the software. The **GATEWAY** is in fact a unique entity that can be retrieved from any scope during execution, thus providing a unified access point to all the information available. This reduces computationally expensive practices like variable duplication and information passing through function arguments.

The real "actors" of an uncertainty quantification problem are contained in the *modules* attached to the *core modules*. Typical examples of *modules* would be, *e.g.*: a generator of random variables distributed according to arbitrary PDFs for the **INPUT** *module*, a FEM simulation tool for the **MODEL** *module*, or a reliability analysis tool for the **ANALYSIS** *module*. The platform allows one to define an arbitrary number of *modules* and select the desired ones at the various stages of the solution of a complex

uncertainty quantification problem. This modularity allows for easy validation of new methods against existing ones, a key feature for research institutions focusing on the development of new uncertainty quantification algorithms. Indeed, new algorithms may be compared with the corresponding reference ones within the same software when the results from different analyses can be kept persistent in memory.

Solving an uncertainty quantification problem with UQLAB

Setting up an uncertainty quantification problem in UQLAB simply consists of creating the desired *modules* with proper configurations. To give a practical example, a generic reliability analysis would require a user to provide the following:

INPUT *module*: a joint PDF representing the uncertain input parameters.
MODEL *module*: a function that operates on samples extracted from the **INPUT** and calculates the corresponding model response;
ANALYSIS *module*: a reliability analysis associated with a failure criterion on the model response.

Once all the ingredients are defined, the analysis is initiated and executed over the defined modules. Note that the modules are independent (black-boxes), which means that it is possible, *e.g.*, to execute repeated analyses with different modelling tools without the need to create new scripts, keeping the results from multiple analyses persistent in memory.

EXTENDIBILITY AND ACCESSIBILITY: A MULTILEVEL COLLABORATIVE DEVELOPMENT MODEL

Due to the unique philosophy of UQLAB, which aims at providing both state-of-the-art UQ tools as well as a powerful framework for the development and validation of new algorithms, its development model has primary role in the design of the software. The target audience of UQLAB is grouped in two main categories, distinguished by their role in the use and/or development of the framework:

end users They are interested in the deployment of the UQ techniques provided by the framework and will not contribute to the extension of the facilities offered by it. They are required to have only minimal programming/scripting skills and are expected to get familiar with UQLAB through existing documented analyses, which they can modify and tailor to their needs. This is the profile of most field engineers and university students;
scientific developers They are trained scientists with relevant expertise in the field of UQ, interested in both exploiting the existing features and in adding new ones. They possess scientific programming skills as well as advanced knowledge of the theoretical framework underneath UQLAB. This is the typical profile of academic researchers or research engineers in the industry.

The development model of UQLab follows this classification, offering a folder-based content management system (CMS) for collaboration-driven and user-contributed code. A researcher (scientific developer) willing to include his/her own code in UQLAB simply needs to copy it in a specific sub-folder of the source tree (including any arbitrary sub-folder structure that he/she may have used) and add a few simple initialization scripts based on existing templates. Upon starting the framework, his/her tool would then be recognized and made available by the framework.

This content-management system, together with the non-intrusive core structure of UQLAB, makes adding plug-ins to existing software simple. This can be achieved by writing elementary wrapper codes that provide the necessary input to the external software, execute it, and parse its results back into the UQLAB framework.

A SIMPLE EXAMPLE

Despite UQLAB still being under development, its modular core has already reached "alpha" stage, and several tools for structural reliability, sensitivity and metamodelling analysis are available within the framework. In this Section, we will use a Monte Carlo reliability analysis of a simple truss structure to showcase some of the features of the framework.

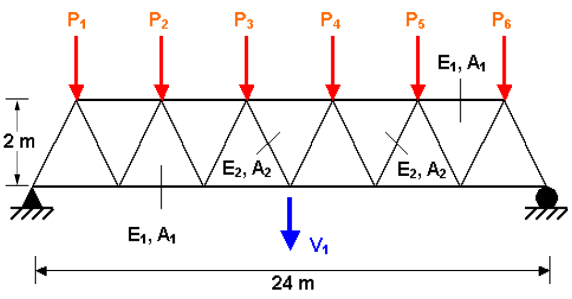
Reliability analysis of a truss structure

A basic "textbook example" of a truss structure under variable load is sketched in Figure 3. On the left panel, the structure is represented with the distributions of the 10 uncertain parameters (the Young moduli, E1 and E2, the beam sections, A1 and A2 and the 6 variable loads, P1-P6). The input variables are distributed according to lognormal or Gumbel distributions, parametrized by their first- and second-order moments. The variables are assumed independent. The failure criterion (*limit state*) in this analysis is defined by a threshold on the displacement at midspan V_1, which is calculated by a black-box in-house developed simple FEM model.

The code that runs the reliability analysis is given on the right panel of Figure 3. The sequence of commands is summarized as:

- The framework is initialized with the `uqlab` command.
- An **INPUT** module is created, based on the provided input-parameter PDFs.
- A **MODEL** module is created with the `uq_create_model` function by specifying that it is contained in an m-file named `'uq_truss'`.
- An **ANALYSIS** module is created by specifying its type (`'uq_reliability'`) and the failure criterion for the midspan deflection point (`'limit_state'`). The method of choice is Importance Sampling (`'IS'`) around the First Order estimate of the failure point (FORM) with a maximum number of samples (`'MaxSamples'`) set to 10^4.
- The analysis is run with the `uq_runAnalysis` command and the results are stored in an appropriate structure within the MATLAB workspace.

Truss structure:



Parameter distributions:

Name	Type	μ	σ/μ
E1, E2 (Pa)	Lognormal	2.1×10^{11}	10%
A1 (m^2)	Lognormal	2.0×10^{-3}	10%
A2 (m^2)	Lognormal	1.0×10^{-3}	10%
P1 - P6 (N)	Gumbel	5.0×10^{4}	15%

$$P_f(V_1 > 0.13\text{m}) = 1.18 \times 10^{-4}$$

code:

```
uqlab
Marg(1).Name = 'E1';
Marg(1).Type = 'Lognormal';
Marg(1).Moments=[2.1e11 2.1e10];
Marg(2).Name = 'E2';
      ...
myInput = uq_create_input(Marg);

Model.Type='mfile';
Model.Function = 'uq_truss';
myModel=uq_create_model(Model);

Analysis.Type='uq_reliability';
Analysis.limit_state = 0.13;
Analysis.Method = 'IS';
Analysis.MaxSamples = 1e4;
uq_create_analysis(Analysis);
uq_runAnalysis;
```

$$P_f(V_1 > 0.13\text{m}) = 1.17 \times 10^{-4}$$

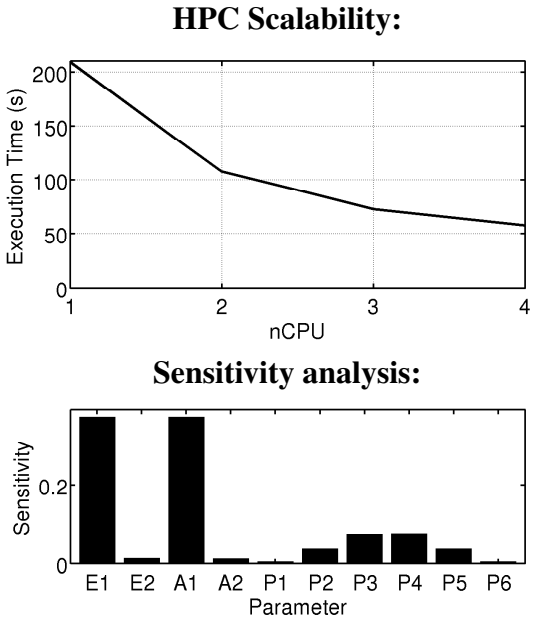
Figure 3. Reliability analysis of a truss structure: problem representation (left) and UQLAB pseudo-code to perform the analysis (right). The results are stored into a structure available in the current MATLAB workspace.

This simple workflow consistently follows the steps described in the global theoretical framework presented earlier.

Advanced features: HPC, metamodelling and sensitivity analysis

In the previous example, the reliability analysis was carried out with an importance sampling algorithm that requires approximately 10^4 model evaluations before converging to a stable result. This was possible due to the simplicity of the model employed in the calculations, requiring only a fraction of a second to execute. Let us now consider a more realistic situation, *e.g.* using a much more time-consuming modelling routine, that would make the evaluation of a large number of samples impossible. In order to make the previous problem solvable, we decided to use an adaptive sparse Polynomial Chaos Expansion (PCE) surrogate model (Blatman and Sudret 2011) evaluated on an experimental design limited to 200 model evaluations. We also want those model evaluations to be automatically distributed on 4 cores of a remote machine. Finally, we also want to perform a sensitivity analysis with respect to the input parameters by calculating the total Sobol' indices from the PCE results. In order to do so in UQLAB, it is sufficient to add the code in Figure 4 to the code in Figure 3, right before the definition of the **ANALYSIS** module.

The code in Figure 4 first defines the parallelization scheme (**DISPATCHER**), by specifying the number of cores to be used (`nCPU`) and a credentials-configuration file (`credentials.txt`), containing information about the remote machine. It then



code:

```
HPCopts.nCPU = 4;
HPCopts.Profile='credentials.txt';
uq_create_dispatcher(HPCopts);

Metaopts.Type = 'uq_metamodel';
Metaopts.MetaType = 'PCE';
Metaopts.ExpDesign.NSamples = 200;
Metaopts.Input = myInput;
Metaopts.FullModel = myModel;
PCmodel=uq_create_model(Metaopts);
uq_calculate_metamodel;

Analysis.Type='uq_sensitivity';
Analysis.Method = 'Sobol';
uq_create_analysis(Analysis);
uq_runAnalysis;
```

Figure 4. Left: scalability and sensitivity analysis results (Sobol' indices based on PC Expansion) for the Truss structure in Figure 3. Right: code used to parallelize and calculate the metamodel and the analysis.

creates a PCE metamodel (`Type = uq_metamodel, MetaType = 'PCE'`) from the `myInput` and `myModel` modules defined earlier (see Figure 3), and sets the size of the experimental design to 200 model evaluations. The metamodel is then calculated (`uq_calculate_metamodel`) and if a valid **DISPATCHER** is found, the calculations of the model responses are automatically parallelized. In order to test the scaling performance of our approach, we designed a FEM routine to require exactly 1s of CPU time for each model evaluation and we ran the metamodel calculation on 1 to 4 cores simultaneously. The results are represented in the top left panel of Figure 4.

Finally, a sensitivity analysis module is created, in perfect analogy with the previous reliability analysis (see Figure 3). The method of Sobol' indices (`Method = Sobol`) is selected for the analysis. Because the current **MODEL** is the newly created PCE metamodel, the global Sobol' indices are automatically calculated from its coefficients (Sudret 2008). The final results of this analysis are shown in the lower left panel of Fig 4. As expected, their symmetry perfectly matches that of the truss model in Fig 3, as well as the reference results given in (Sudret 2007).

CURRENT STATE OF THE FRAMEWORK AND OUTLOOK

After approximately one year of development, UQLAB has reached the "alpha" stage, and is currently being tested internally by both researchers and students. We anticipate to enter a "close beta" stage in the first quarter of 2014, with a first public release date

still to be defined.

At the current stage of development, the "core structure" of the framework, the content management system and a number of wrappers for existing modelling and meta-modelling software have been implemented and thoroughly tested.

In terms of scientific features, the following modules are currently available:

- **INPUT** module: a large number of marginals with elliptic copula correlation (Gaussian and T-Student).
- **MODEL** module:
    - Support for user-provided string functions.
    - Support for m-file based models (MATLAB functions).
    - Templates for implementing proprietary code wrappers.
    - Support for directly extracting data from binary/text files.
    - Advanced metamodelling module:
        - Kriging (ordinary, linear, universal and user defined regression trends; support for several autocorrelation kernels, local and global hyperparameter optimization including gradient-based, genetic and hybrid genetic + gradient based algorithms; maximum likelihood and cross-validation objective functions).
        - standard and sparse Polynomial Chaos Expansion (projection methods: standard and sparse (Smolyak') Gauss quadrature; adaptivity in the polynomial degree and sparse basis regression methods, *e.g.*, Least Angle Regression (LAR); simple experimental design enrichment).
        - automated HPC distribution of experimental design calculations (common to all metamodelling methods).
- **ANALYSIS** module:
    - Structural Reliability module:
        - Support for user defined limit-state functions.
        - First- and Second-order analysis (FORM and SORM).
        - Monte-Carlo reliability analysis.
        - Importance Sampling.
    - Sensitivity Analysis module:
        - Sobol' indices.
        - Elementary effects analysis.

For each of the available modules, an extensive library of test scripts and real-life examples that gradually build up in complexity are provided, to allow users to familiarize themselves with the UQLAB facilities.

A tight schedule for the development of additional standard algorithms as well as advanced features and enhancements to the platform has been set for the coming

months at the Chair of Risk, Safety and Uncertainty Quantification in ETH Zurich. These include: support-vector machines metamodelling, subset-simulation, additional interfaces to commonly available HPC infrastructures, visualization tools, a graphical user interface and a large library of plug-ins to third party modelling/UQ codes.

CONCLUSIONS

We successfully implemented a software framework based on the global uncertainty quantification framework described in (Sudret 2007; DeRoquigny 2008). With its innovative and simple design philosophy as well as its collaborative development model, it is well suited to encourage both the academic and the industrial R&D research communities to employ and further develop state-of-the-art uncertainty quantification algorithms. In the coming years, the set of features it offers will be substantially increased to include most of the latest developments in the rapidly evolving field of uncertainty quantification.

REFERENCES

G. Andrianov, Burriel, S., Cambier, S., Dutfoy, A., Dutka-Malen, I., de Rocquigny, E., Sudret, B., Benjamin, P., Lebrun, R., Mangeant, F. and Pendola, M. (2007). "Open TURNS, an open source initiative to Treat Uncertainties, Risks'N Statistics in a structured industrial approach." *Proceedings of the ESREL'2007 Safety and Reliability Conference*, Stavenger: Norway.

Blatman, G. and Sudret, B. (2011). "Adaptive sparse polynomial chaos expansion based on Least Angle Regression." *J. Comput. Phys.,* 230, 2345-2367.

Bourinet, J.-M., Mattrand, C. and Dubourg, V. A. (2009), "Review of recent features and improvements added to FERUM software", *Proceedings of the 10th International Conference on Structural Safety and Reliability, ICOSSAR'09*.

De Rocquigny, E., Devictor, N. and Tarantola, S. (2008). "Uncertainty in industrial practice – A guide to quantitative uncertainty management." *John Wiley & Sons*.

Eldred, M.S., Giunta, A.A., van Bloemen Waanders, B.G., Wojtkiewicz, S.F., Jr., Hart, W.E., and Alleva, M.P. (2004). "DAKOTA, A multilevel parallel Object-Oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis. Version 3.3 reference manual." *Sandia Technical Report SAND2001-3515*.

Schuëller, G.I. (Editor) (2006). "General-purpose softwares for structural reliability analysis." *Struct. Saf.,* 28.

Sudret, B. (2007). "Uncertainty propagation and sensitivity analysis in mechanical models - Contributions to structural reliability and stochastic spectral methods." *Hab. à diriger des recherches, Université Blaise Pascal, Clermont-Ferrand, France*.

Sudret, B. (2008) "Global sensitivity analysis using polynomial chaos expansions." *Reliab. Eng. Sys. Safety,* 93, 964–979.