Ingredients for an innovative uncertainty quantification platform in MATLAB

Stefano Marelli & Bruno Sudret
Chair of Risk, Safety and Uncertainty Quantification, Institute of Structural Engineering, ETH Zürich
marelli@ibk.baug.ethz.ch, sudret@ibk.baug.ethz.ch

Abstract: Uncertainty quantification is an emerging field in computer simulation-based scientific applications. Its general formulation covers a vast field of approaches including, e.g., structural reliability, sensitivity analysis, reliability-based design optimization and Bayesian techniques. The UQLAB project aims at developing a MATLAB-based framework that could enable researchers and field engineers to both use and develop new uncertainty quantification algorithms in several types of distributed computing environments. Ease of use, extendibility and handling of non-intrusive stochastic methods are core elements of its development philosophy. The platform comprises a highly optimized core probabilistic modelling engine, a simplified API layer that provides unified access to heterogeneous high performance computing resources and a content management system that allows users to develop additional custom modules to suit their needs. In this contribution we intend to present the global architecture and to demonstrate some of the core capabilities of the UQLAB framework in its early stages of development.

Keywords: Uncertainty Quantification, Global Uncertainty Framework, UQLab, Structural Reliability, Matlab

1 Introduction

Uncertainty quantification through computer simulation is an emerging field at the boundary between computer simulation-based engineering and applied mathematics, statistics and probability theory. Broadly speaking, it aims at identifying sources of uncertainty in each of the components of the simulation of physical quantities and propagating this uncertainty into the model responses. This is a general formulation that covers a vast field of approaches
including, among others, structural reliability, sensitivity analysis, reliability-based design optimization and Bayesian techniques for calibration and validation of computer models.

In terms of research, contributions to this broad topic equally come from the engineering, statistics and applied mathematics communities, sometimes with their own vocabulary. In terms of computational tools, only a few are readily available (e.g. free software like FERUM [2] and OpenTURNS [1], or commercial ones like COSSAN and Nessus). A review on structural reliability software is available in [5]. None of them, however, covers the broad scope mentioned above.

Furthermore, to our knowledge none of the existent software is designed to be easily extended by the engineering research community. The use of powerful but complex languages like C++ and Python as well as the widespread adoption of Object-Oriented programming at all levels of development may discourage the scientific developers to make use of the facilities provided by existing software and opt instead for in-house re-development of yet more heterogeneous codes.

In an attempt to overcome such limitations, the Chair of Risk, Safety and Uncertainty Quantification in ETH Zürich has started the UQLAB (Uncertainty Quantification in MATLAB) project, aiming at the development of a powerful, modular and simple to extend software framework for uncertainty quantification in Engineering.

The main defining goals of the UQLAB project can be summarized as:

- providing a complete set of tools for uncertainty quantification in engineering applications;
- ensuring ease of use for both academic students, researchers and field engineers;
- designing a modular structure easy to extend by not highly-IT-trained scientists;
- providing high interoperability with existing third party software in a non-intrusive, “black box”-type approach;
- easing the deployment of uncertainty quantification algorithms on a variety of high-performance computing platforms.

In order to achieve all of the mentioned goals, our software architecture eventually converged to a MATLAB based highly modular software framework.

2 A global uncertainty quantification framework

Uncertainty quantification in civil and mechanical engineering is a vast scientific research topic that has seen active development for more than 40 years. Due to the broad spectrum of its possible applications, uncertainty quantification is a highly heterogeneous field. Therefore, in order to build an uncertainty quantification software with a general enough scope, a correspondingly general theoretical framework is required. At the backbone of the architecture of the UQLAB software lies the global uncertainty framework developed by SUDRET (2007) and reviewed, e.g. in [7][3], that is sketched in Fig. [1]
Fig. 1: Visual representation of the global theoretical framework for uncertainty quantification developed by [7] and [3], which gives the theoretical foundation to the UQLAB software.

According to this framework, several common steps need to be undertaken in every uncertainty quantification problem:

- **Step A** consists in defining the physical model and the quantities of interest on which the analysis will be performed (e.g., a failure criterion). It is a deterministic representation of an arbitrarily complex physical model (e.g., a finite element model).

- **Step B** consists in identifying, quantifying and representing the sources of uncertainty in the system. Such quantities will serve as the input for the modelling in Step A, and are usually represented by a vector of random variables whose joint probability density function (PDF) is defined on the basis of the available information and data.

- **Step C** consists in the propagation of the uncertainty in the input random variables defined in Step B through the modelling in Step A, thus evaluating the resulting uncertainty in the model response.

- **Step C’** is an optional step that consists in exploiting by-products of the analysis in Step C in order to rank the sources of uncertainty according to their impact onto the quantities of interest. This step is known as sensitivity analysis.

Each problem of uncertainty quantification can be solved by following this approach, as long it can be decomposed in three main components: physical model, input variables, uncertainty propagation analysis. These three components introduce a semantic distinction between the actors involved in any uncertainty quantification problem. This theoretical framework, therefore, provides the ideal foundation to the development of the information flow model in a multi-purpose uncertainty quantification software.
3 From a theoretical framework to its implementation

3.1 The software architecture

To achieve the same level of generality and flexibility of the theoretical framework presented in Section 2, 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 creation of content, rather than on the content itself;
- its development model plays a relevant role in its own architecture;
- its features can be arbitrarily extended, as long as they can be represented within its structure.

The core architecture of the UQLAB uncertainty quantification framework closely follows the semantics defined in the previous Section by implementing a modular, partially hierarchical structure sketched in Fig. 2. The three steps identified in Fig. 1 directly map to

![Diagram](image)

Fig. 2: The modular core at the heart of the UQLAB framework architecture. At any stage of the analysis, an arbitrary number of elements can be connected.

supermodules represented as light shaded boxes in Fig. 2: Step A (physical modelling) is mapped to the MODEL, Step B (sources of uncertainty) to the INPUT and Step C (uncertainty propagation) to the ANALYSIS supermodules, respectively. Auxiliary supermodules (e.g. DISPATCHER and WORKFLOW in Fig. 2) similarly represent additional actions that need to be handled during a calculation (e.g. dispatching calculation on HPC resources).

Each supermodule has several connections: a single connection to a central unique GATEWAY (the central circle in Fig. 2), and an arbitrary number of connections to children mod-
ules (represented by grey boxes in Fig. 2). The GATEWAY is a unique entity that can be retrieved at any time and from any location within the framework, with the function of providing a unique access point to every supermodule, thus making the information ubiquitous throughout the framework. Finally, each module connected to a supermodule represents an actual “entity” belonging to it. To give a few examples, a module connected to the ANALYSIS supermodule could be the “reliability analysis” module, while modules connected to the MODEL supermodule could be, e.g. an analytical function or a finite element model.

Implementing an uncertainty quantification analysis within the framework, therefore, would consist in defining the necessary modules and their configuration for each of the supermodules in Fig. 2. To give a practical example (full pseudo-code will be given in Section 5), a Monte-Carlo reliability analysis of a simply supported beam would require the user to define the following modules:

- an INPUT module: a set of probability density functions representing the uncertainty in the beam geometry and applied loads. Dependent variables may be defined using the copula formulation [6];
- a MODEL module: the midspan deflection of a simply supported beam as a function of the variables defined in INPUT;
- an ANALYSIS module: a Monte Carlo reliability analysis associated with a failure criterion related to the maximal admissible deflection.

Once all the ingredients are defined, the analysis is started and it is executed over the defined modules. It should be noted, however, that an arbitrary number of modules can be connected to each supermodule in Fig. 2. Therefore, in analogy with the possibilities offered by the theoretical framework in section 2, it is possible to combine different module types for different uncertainty quantification problems.

The simplest case in which this property may be desirable is that of a validation of a new modelling method. Let us imagine that in the previous example we decided to test a newly developed modelling strategy that makes use of, e.g., a complex FEM scheme to accurately predict the beam midspan deflection, and that we wanted to compare it with results based on the analytical solution of the problem. It would then be sufficient to add a new module to the MODEL supermodule based on the new FEM modelling routine and to define a new workflow that makes use of the same INPUT and ANALYSIS, but replaces the old MODEL module with the new one.

Handling multiple workflows is exactly the purpose of the WORKFLOW supermodule depicted in Fig. 2. In practice, a user can define an arbitrary number of workflows that can be executed sequentially over the course of an uncertainty quantification analysis. The importance of this facility becomes clear when solving highly complex uncertainty quantification problems on shared high-performance-computing platforms: surrogate modelling, higher-order reliability methods and other sophisticated techniques can be streamlined in a set of sequential workflows that can be queued and executed in a single HPC job, minimizing repeat calculations and queue waiting times. Another typical application example of the workflow feature is the case when the impact of different probabilistic INPUT should be compared.
Starting from the existing Input 1, a new Input 2 may be built up, and a new workflow accordingly.

The final supermodule in Fig. 2 is the DISPATCHER. Once again semantically separated from the other supermodules, the DISPATCHER aims at offering a unified interface to a set of heterogeneous High Performance Computing facilities via the use of simple configuration cards. When requesting the execution of an analysis, the dispatcher would appropriately select the actual commands that need to be executed to perform the analysis on the correct infrastructure. As an example, the default DISPATCHER is the “local” one, which simply executes the program locally without modifications. A more complex example (but perhaps more useful in a real world scenario), is the “ssh+cluster+queuing system” dispatcher, that first connects to the configured computing resource (cluster) via the specified connection type (ssh), copies the data necessary to the execution to the remote host, creates any job scripts necessary for its execution (queuing system), submits them and finally retrieves and merges the results when the computations are done. As arbitrarily complex as those operations may be, they are semantically completely detached from the other aspects of the computational problem, therefore they have been included into a separate supermodule.

3.2 The MATLAB implementation

One of the defining goals of our endeavour in developing UQLAB (see Section 1) is making both the use of existing and the development of new uncertainty quantification algorithms easily accessible to not-IT-trained professionals (e.g. students and field engineers). Due to its widespread use in the engineering community, its quick learning curve, its portability and the advanced programming interface it offers, we adopted MATLAB as the programming language for our framework.

The core structure of the UQLAB software presented in Fig. 2 is coded in an object-oriented fashion to preserve the semantics defined in Section 2. The GATEWAY is a MATLAB implementation of the singleton design pattern, guaranteeing its uniqueness and retrievability at any point in the MATLAB session, by means of a single protected global variable (its instance handle). The supermodules are created as soon as an instance of the GATEWAY instance is created, and they are linked with it. Each of the supermodules is in turn an object that only contains information about the existent modules, as well as methods to add, retrieve and remove them. Due to their nature, the GATEWAY and the supermodules are lightweight entities that introduce negligible overhead. In particular, retrieving the current gateway and the pointers to the existing module handles can be done at the cost of a handful of memory operations within any scope of a MATLAB session. High performance, stability and small overhead are extremely important in the design of these components, as they are ubiquitous throughout the execution of any software in the framework.

Each module created by a supermodule is by itself an object that contains all of the information needed to perform the task it is in charge with. As an example, a module created by the INPUT supermodule would contain all the information about the probability distributions that need to be sampled in the analysis, as well as a method that extracts a sample from them. Analogously, a module created by the MODEL supermodule would contain any necessary runtime parameter, as well as a method to run the modelling routines on the current
input.

Due to this highly modular structure and its object oriented nature, diagnostic tests can be executed at each module creation, or by means of more advanced infrastructures provided by MATLAB, like event listeners and callbacks. Therefore, a healthy degree of internal coherency in the framework can always be guaranteed at any stage of its execution.

This high-level functionality is exposed to the users by means of a simple command line interface (CLI) and a more advanced programming interface (Application Programming Interface), neither of which requires any knowledge of object-oriented programming. Examples of both interfaces will be presented in Section 5.

The conceptual separation between actors in an uncertainty quantification introduced in Section 2 has an important property: the uncertainty quantification framework is intrinsically non-intrusive. Non-intrusiveness guarantees that, should any of the modules be substituted by a “black box” equivalent, all of the remaining modules would remain unaffected. This is a highly desirable property in a framework that aims at unifying existing software tools with different technical requirements and input-output formats.

4 Extendibility and accessibility: a multilevel collaborative development model

The collaborative nature of the UQLAB project, has relevant consequences on the design model of the platform. The audience of the software, intended as both final users and scientific developers/collaborators, has been grouped in three categories, distinguished mainly by their role in the use and/or development of the framework:

- **end users**: users that will not contribute to the extension of the facilities offered by the platform, therefore interested mostly in the deployment of existent techniques. They are not required to possess any particular programming skill, only the basic knowledge of the theoretical framework (see Fig. 1). It is expected that these users learn how to use 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**: trained scientists with relevant expertise in the field of uncertainty quantification, interested both in taking advantage of the existing features and in testing and adding new algorithms into the platform. They possess scientific programming skills as well as the knowledge of the theoretical framework. It is the typical profile of the academic researcher;

- **core developers**: scientists highly trained in IT and high performance programming, mainly concerned with the development of the core routines that provide access to the platform contents, as well as with the optimisation of contents provided by other scientific developers. They possess advanced programming skills in Object-Oriented MATLAB and other languages, as well as knowledge of system architecture and related fields. This is the profile of a small number of IT-professionals that are concerned with the “software” perspective of the the UQLAB framework development.
Fig. 3: The folder-based content management system of UQLab at the contrib stage. Note how, provided the custom_model_run.m and custom_model_initialize.m scripts exist, the sub-folder structure of new method is completely arbitrary.

The development model of UQLab is designed following this classification, offering a folder-structure-based, content management system (CMS) for collaboration-driven and user-contributed code (Fig. 3), that makes integration of new features or improvements in the existing code-base simple.

A researcher (scientific developer) willing to include his newly discovered “new method” of modelling in the framework, could simply operate as follows:

- he can copy his own code into the “external” sub-folder of the MODEL supermodule source tree node, under the name custom_model. Extra additional MATLAB scripts can be added in the same folder, with a suitable naming convention (e.g. custom_model_initialize.m or custom_model_run.m), that perform additional consistency checks/initializations/operations;
- upon restart, the framework recognizes the existence of a new MODEL type and makes it available for use;
- the GATEWAY and all the supermodules are made available ubiquitously at any point of his code with a very simple set of API functions, allowing for a very fast integration;
- as soon as the “new method” is tested thoroughly, it can be moved to the “contrib” sub-folder of the MODEL source tree and committed to the main code repository. This is the stage depicted in Fig 3;
- upon functionality verification from the core developers, the module is now available as an optional module to the end-users, who are free to enable/disable contrib codes at will.
- finally, if the end-user feedback is positive enough, the new method may undergo a thorough review and optimization from the core developers, and finally be moved into the builtin sub-folder of the MODEL source tree node, to be shipped into the next public release of the framework.

This simple content management system, together with the highly non-intrusive core structure of UQLab (see 3.1), make it very simple to add plug-ins to existing software without
the need to change it in any way. This is achieved by writing simple wrapper codes that provide the necessary input to the external software, execute it, and parse its results back into the UQLAB framework.

The UQLAB framework will be released as an open-source project under the terms of the GNU Lesser General Public License (LGPL), therefore promoting its further development from both the academic research community and industrial R&D environments.

5 A simple example

Although UQLAB is still in its early stages of development, its modular core has already reached pre-alpha stage (stable for demonstrative purposes), and a basic set of tools for structural reliability analysis has been implemented within the framework. In this Section we will use a Monte Carlo reliability analysis of a simply supported beam to showcase some of the features of the framework.

5.1 Reliability analysis I: the end-user perspective

The basic “textbook example” of reliability analysis of a simply supported beam under uniform load is sketched in Fig. 4.

![Pseudocode](uqlab)

\[
V = \frac{5}{32} \frac{pL^4}{Ebh^3}
\]

Fig. 4: Reliability analysis of a simply supported beam: problem representation (left) and pseudocode in UQLAB that performs the analysis (right). The results will be stored into a `uq_results` variable in the current MATLAB workspace.

On the left panel the sketch of the beam and its uncertain parameters (the uniform Young’s modulus \(E\), the beam length, thickness and height, \(L, b\) and \(h\), respectively, and the uniform load \(p\)) are given. Each of the input variables is defined by a normal (or lognormal) distribution identified by its first and second order moments. The variables are assumed independent. The failure criterion in this reliability analysis is defined as a threshold on the displacement at midspan \(V\), which has the simple analytical expression given under the beam scheme in Fig. 4. On the right panel the pseudo-code necessary to run the analysis within the framework for an end-user (see Section 4) is given. The sequence of commands is commented in
the following:

- The framework is initialized with the uqlab command.
- A set of input distributions is defined through their moments in the “Marginals” structure. This structure is then passed as an argument to the uq_create_input function, which creates the appropriate INPUT module with the specified properties. Note that the module is also given a text identifier (“myInput”) that will uniquely identify it when defining multiple workflows (cf. Section 3.1).
- In full analogy, a MODEL module is created with the uq_create_model function by specifying that it is a simple function type, and that the function that needs to be executed is “SSbeam”. It is given the text identifier “myBeam”.
- An ANALYSIS module is finally created by uq_create_analysis, and named “myAnalysis”. It is a “Reliability”-type analysis, to be performed with the “Monte Carlo” method. A value for the failure threshold is also specified.
- Finally, the analysis is run with the uq_run_analysis command, and the results are stored in an appropriate variable within the MATLAB workspace.

This simple workflow accurately follows the theoretical framework described in Section 2 and sketched in Fig. 1. The results of this analysis for a realistic set of input parameter distributions and $10^9$ Monte Carlo iterations are shown on the left panel of Fig. 5.

![Fig. 5: Left: sample from the $10^9$ calculated displacements from the Monte Carlo reliability analysis described in Section 5.1. Right: corresponding scalability plot on a remote cluster.](image)

Distributing the calculation on several cores on a remote machine simply requires the addition of the following lines to the pseudo-code in Fig. 4, before the uq_run_analysis line:

```matlab
HPCOpts.configCard='ssh+torque+cluster';
HPCOpts.nCPU=N;
uq_create_hpc_dispatcher('myDispatcher',HPCOpts);
```

This simple additional configuration allows one to build scalability performance analyses like the one represented on the right panel in Fig. 5. The target computational facility was in this study an 8-core shared-memory remote system with a Torque queuing manager set-up to mimic most of the HPC facilities largely available in both academic and industrial research.
5.2 Reliability analysis II: the scientific developer perspective

The simple interface for the end-users was outlined in the previous section, but an equally important goal in the UQLAB development philosophy is its ease of extendibility. In this Section we illustrate how a scientific developer can create his own analysis routines by taking advantage of the facilities offered by the UQLAB framework.

The pseudo-code that implements the core Monte-Carlo routine in the simply supported example in Fig. 4 is shown in Fig. 6.

```matlab
function results = MonteCarlo
% retrieve the framework: several variables will be created
% in the workspace: UQ_input, UQ_analysis and UQ_model
uq_retrieve_session;

% retrieve the analysis parameters (failure threshold
% and number of samples) from the framework
failure = UQ_analysis.failure;
nsamples = UQ_analysis.nsamples;

% evaluate the model response on a sample from the current input
% of the retrieved sample size
displacements = uq_get_model_response(nsample);
totfailures = length(find(displacements >= failure));

% store the failure probability in the framework’s results array
UQ_analysis.results{end}.failure_probability = totfailures/nsamples;
```

Fig. 6: Pseudo-code of the simple Monte-Carlo reliability analysis used in Section 5.1 and Fig 4. Note that `uq_get_model_response` will retrieve which MODEL needs to be evaluated and which INPUT automatically from the current WORKFLOW module (cf. Fig. 3.1).

The crucial interaction point with the underlying framework is provided by the `uq_retrieve_session` helper function. It is a lightweight function that retrieves all the pointers to the core elements in Fig. 2 and to the currently selected modules, and conveniently exposes them to the user in the form of a set of variables `UQ_input, UQ_analysis, etc`. An additional set of helper functions like `uq_get_model_response` (that automatically evaluates the currently selected MODEL module on a sample from the currently selected INPUT module) are available to the developer as a part of the scientific developers programming interface. A comprehensive reference to such functions is outside the scope of this paper and is currently being rapidly extended.

As it is clear from the pseudocode in Fig. 6, the amount of code rework that is necessary to include a script within the UQLAB framework is minimal, de-facto only substituting the standard information-passing practices (e.g., function arguments, global variables) with a single call to `uq_retrieve_session`, and the input-sampling and modelling stages with the proper `uq_*` helper functions.

6 Current state of the framework and outlook

At its current stage of development, only 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 to a satisfactory degree and thoroughly tested. A tight
schedule for porting and rewriting a number of other existent uncertainty quantification algorithms has been set in place for the coming months, and will be developed in the short term by the members of the Chair of Risk, Safety and Uncertainty Quantification at ETH Zürich. As soon as a minimalistic set of techniques will be available and well documented, a first public release of the UQLAB software will be published, and an effort to start a collaborative development stage with other research institutions will be initiated.

Foreseen features will include reliability analysis à la FERUM [2], meta-modelling techniques (polynomial chaos expansions [7], Kriging [4], support vector machines), sensitivity analysis (Sobol indices), Markov Chain Monte Carlo methods, etc.

7 Conclusions

We successfully implemented a software framework based on the global uncertainty quantification theoretical framework described in [7, 3]. With its innovative design philosophy and 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 to end-users will be substantially increased to include most of the latest developments in the rapidly evolving field of uncertainty quantification.

Literature


