Available online at www.eccomasproceedia.org
Eccomas Proceedia UNCECOMP (2021) 209-217

ECCOMAS
Proceedia

UNCECOMP 2021
4th ECCOMAS Thematic Conference on
Uncertainty Quantification in Computational Sciences and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou (eds.)
Streamed from Athens, Greece, 28 -30 June 2021

UNCERTAINTY QUANTIFICATION IN THE CLOUD WITH UQCLOUD

Christos Lataniotis[1], Stefano Marelli[1], and Bruno Sudret[1]

[1]Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich
Stefano-Franscini-Platz 5, 8093, Zurich, Switzerland
e-mail: {lataniotis,marelli,sudret}@ibk.baug.ethz.ch

Abstract. *General-purpose uncertainty quantification software has become a well established requirement in modern engineering workflows. Different communities (e.g. applied maths, engineering, economics, etc.), however, generally employ diverse arrays of technologies and workflows, from computing infrastructure to programming languages. To overcome the intrinsic limitation of a single language, standalone software package, we introduce* UQCLOUD*, an OS- and programming language- independent, cloud-based version of* UQLAB*.* UQCLOUD *follows a software-as-a-service (SaaS) model, that allows anyone to take advantage of the well-established* UQLAB *suite, without the need to adapt their computational workflows to include* MATLAB*.*

Keywords: Uncertainty Quantification, UQLab, UQCloud

ISSN:2623-3339
Peer-review under responsibility of the organizing committee of UNCECOMP 2021.
doi: 10.7712/120221.8033.18990

1 INTRODUCTION

Including uncertainty quantification (UQ) in modern research and engineering practice has become a well established trend in the past decades. From academic research to *validation, verification and uncertainty quantification* (VVUQ) in engineering design [8], dealing with uncertainty is becoming a standard requirement, even for regulatory bodies. This trend has been enabled by the constantly increasing availability of accurate computational models and related high performance computing infrastructure, and at the same time by recent developments in general-purpose uncertainty-quantification software. A remarkable offering of the latter is available at the time of this writing, from the well-established C++-based Dakota project in the United States [3], to the PYTHON-based Open-TURNS in France [1], the MATLAB-based COSSAN in the UK [10], and the more recent, once again PYTHON-based [4] and [9], to mention a few. Along the same lines, the MATLAB-based UQLAB software framework [5] began its design and development phase back in 2013, with the goal of providing a state-of-the-art uncertainty quantification software that would be accessible to applied scientists of all backgrounds, regardless of their programming experience. Given its widespread adoption, counting over 3,500 unique registered users worldwide since July 2015, it is clear that both its set of features and its intuitive, beginner-friendly interface have hit a sweet-spot in our intended audience.

As an effort to further disseminate the adoption of advanced UQ tools across disciplines, in 2019 we launched the UQWORLD online community ([6], `https://uqworld.org/`), which now counts hundreds of users worldwide. Among the feedback collected by several users on that platform , what is commonly seen as a deterrent in the adoption of UQLAB in a number of communities, especially outside engineering, is the programming language choice. The classical "standalone software" paradigm, poses an important limitation in its adoption, especially in industrial contexts: it can be difficult to incorporate it into existing workflows that make use of different software.

Increasing portability outside the pure cross-OS compatibility offered by MATLAB is not an easy task to achieve. UQLAB has reached a remarkable degree of maturity, with over a hundred thousands lines of optimized code that in many cases capitalize on the efficient linear algebra facilities offered by MATLAB. Porting the entire software to a different language would therefore be complex both from a technical perspective (significant expertise would be needed in both the underlying theoretical tools and in the programming language of choice), and from a practical one (it may result in actually slower results, and it would create a maintenance nightmare). UQLAB is also an active project, constantly evolving at its own pace thanks to the work of the researchers and developers from the Chair of Risk, Safety and Uncertainty Quantification in ETH Zurich. Adding to this the rapidly evolving landscape of scientific programming languages (e.g., PYTHON 3 vs PYTHON 2.7, the rise of Julia, the prevalence of C in certain modeling communities), makes it clear that any port to a specific language would only be a temporary patch to fix a greater underlying issue.

With this in mind, we started exploring modern alternatives to the classical standalone software model, that could provide an OS- and programming-language- agnostic, portable version of UQLAB. In this contribution we present UQCLOUD, a modern *software-as-a-service (SaaS)* incarnation of UQLAB that aims at solving the three key aspects of portability, continuous integration with the main UQLAB software, and performance.

In this paper we present a brief review of the state of the UQLAB ecosystem, and then introduce the main structure of UQCLOUD, with an outlook to its implications. To showcase the advantage of such an infrastructure, we also showcase UQ[PY]LAB, a set of software bindings

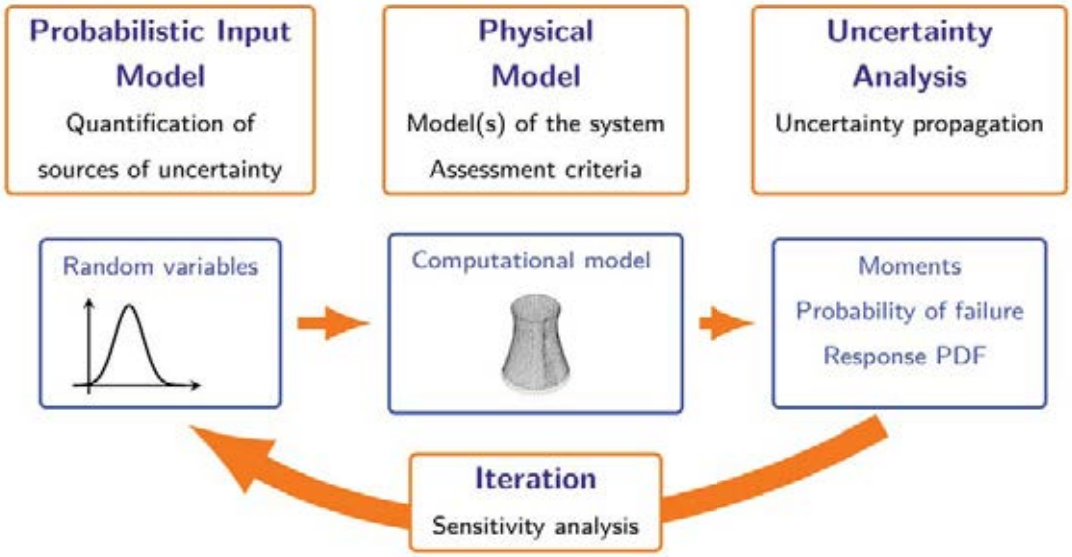


Figure 1: The general UQ framework for managing uncertainty (after [12, 2]) that defines the UQLAB semantics.

that provide full access to UQLAB in PYTHON.

2 THE UQLAB ECOSYSTEM

2.1 The UQLab software framework

At the core of the UQLAB project, lies the UQLAB software framework (`www.uqlab.com`), initially released in 2017 after a two years-long beta testing phase. Designed around the general framework for UQ originally developed in [12, 2] and illustrated in Figure 1, UQLAB provides a natural sandbox for the design and development of complex UQ analyses, without requiring extensive technical knowledge from the users. As of March 2021, with over 3,500 unique registered users from 90 countries worldwide, UQLAB has become a household name in the UQ software community.

In terms of software and development model, UQLAB has reached the maturity stage. At the same time, its wide adoption in largely different fields (ranging from theoretical physics to medical engineering, from computational macroeconomics to disease propagation) is a testimony to the growing need of a software designed around perspective users, rather than just around the profiles of the developers. Its self-imposed target of disseminating UQ outside the bounds of traditionally UQ-savvy fields has been successfully maintained in the past few years.

UQLAB is a public mirror to the research conducted at the Chair of Risk, Safety and Uncertainty Quantification at ETH Zurich (`www.rsuq.ethz.ch`). Its scientific content (embedded in the open source UQLABMODULES) keeps therefore growing at a steady pace, and it is not expected to relent any time soon.

2.2 UQWORLD: a global UQ community

While on the one hand the rapid diffusion of UQLAB demonstrated that it helped filling the niche of general-purpose, accessible UQ software, it also highlighted a second need in the scientific community. Since its early stages, the UQLAB userbase has been highly varied in the degree of technical skills, academic background, and applications. No single resource, however, was available to group together such a large melting pot of different professionals. In

Figure 2: The UQWORLD homepage

2019, the UQWORLD online community [6] was created, with the goal of providing an open forum for UQ practitioners and experts to get in touch and discuss about their needs, or share their expertise.

While a section of the community website (see Figure 2) is dedicated to the use of UQLAB, the vast majority of the available resources are software-independent, and as of today several hundreds of users interact on the community forums, seeking for answers and providing feedback and suggestions. While still in its infancy, UQWORLD is rapidly gaining momentum, and it provides a good starting point to get a pulse of the needs of UQ practitioners worldwide.

A UQLAB-related topic that comes up the community discussions (see, e.g., `https://uqworld.org/t/matlab-but-but-python-r-c-cobol/147`), is the choice of developing UQLAB in MATLAB. Many community members felt the choice of MATLAB somewhat exclusive, as it does not cater to the entire userbase of UQLAB, something that is essentially impossible due to its sheer variety. With this in mind, we started brainstorming about a possible solution to this issue, that would not require translating UQLAB in a number of different languages, each with their own strengths and weaknesses, and highly specialized syntax. The result is UQCLOUD, a modern take on software that is independent on the user platform.

3 INTRODUCING UQCLOUD

3.1 A paradigm shift: from local software, to cloud-based software-as-a-service

UQCLOUD aims at bridging the gap between the UQLAB software and any development or production environment that does not include MATLAB. Due to the numerous challenges involved in translating and maintaining several UQLAB variants in different programming languages, UQCLOUD adopts a more modern software-as-a-service (SaaS) paradigm. In a nutshell, it provides an online programming interface (API) that provides UQ computations on demand, by relying on a UQLAB-powered service running on one or more cloud computing instances. Because the UQLAB service is a deployed program created through the MATLAB deployment toolbox, it is itself, as a matter of fact, MATLAB independent.

The UQCLOUD platform consists of several instances that are deployed on cloud premises. An abstract view of the components of such an instance is shown in Figure 3. Each box denotes

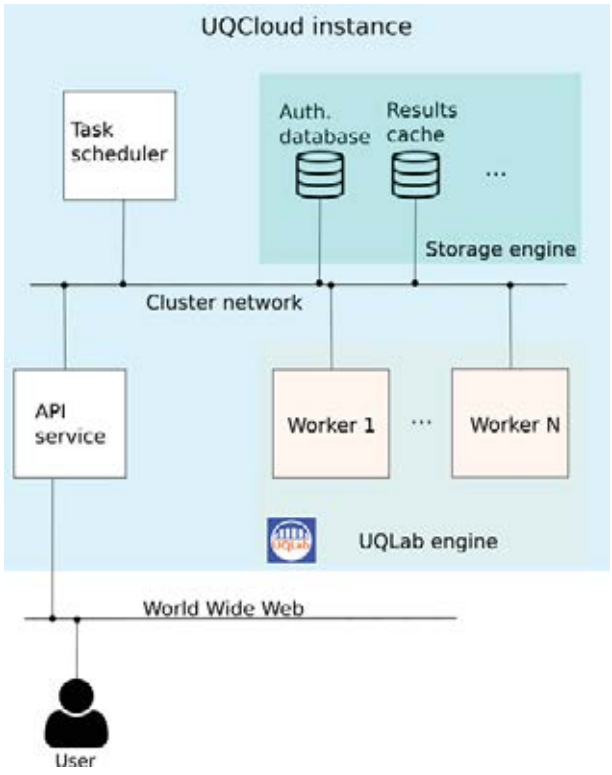


Figure 3: An abstract view of the components of a UQCLOUD instance.

a containerised computing environment [7]. Such environments are virtual lightweight replicas of separate computing nodes, each performing a predefined set of tasks.

At the core of UQCLOUD lies the so-called UQLAB *engine*. It consists of several containers, the *workers*. Each worker is designed to execute any UQLAB-related computation that it is tasked with. To ensure licensing compliance, no MATLAB session is running in any component of UQCLOUD. Rather, a standalone UQLAB-based custom designed executable server is deployed through the MATLAB Compiler™[1]. The workers require therefore no MATLAB installation (and associated licensing).

In practice, users do not directly interface with the UQLAB workers. Instead, they contact the API service container, either directly or through dedicated software bindings. The communication follows the secure industry-standard REST API paradigm (see e.g. [11]). The role of this service is to receive computation requests from the users and return the results in an asynchronous manner. Only authenticated users can submit such requests, which serves two purposes: (1) access control to cloud computing resources, and, (2) performing each UQ computation within a persistent UQLAB session that is dedicated to that user.

The latter is a major requirement for the core UQCLOUD offering, which aims to deliver virtually all the features provided by UQLAB. In this setting, users can perform arbitrary workflows regardless of their complexity. This is made possible by an underlying persistent, user-specific UQLAB session on the cloud that can contain several re-usable INPUT, MODEL and ANALYSIS objects similar to a native UQLAB session within MATLAB.

Finally, the communication between the API service and the workers is mediated by a task-scheduler, that deals with the distribution and bookkeeping of user tasks across workers, and giving appropriate responses to user queries through the API service.

[1] `https://mathworks.com/products/compiler.html`

Such an infrastructure also enable full bi-directional asynchronous communication with the client. In other words, this provides support for workflows that require computational model evaluations outside the cloud (e.g. local). This is relevant, e.g., when local software is required to perform an analysis, or in the case of strict software licensing. As an example, users can perform reliability analysis on the cloud, but every limit state function evaluation can be executed on either local or dedicated hardware. This is even supported in adaptive settings, where continuous transactions between the user and UQCLOUD are needed to identify the most parsimonious solution path to the problem.

Although one can communicate with UQCLOUD directly through so-called REST API calls (structured messages in JSON format), this type of communication is recommended only for very advanced usage, e.g. to interface existing software to UQCLOUD. In general, this lack of user-friendliness is expected to deter most of the users with limited expertise in implementing such calls. Hence, as an integral part of UQCLOUD, we are also developing user-facing software bindings. Those are language-specific packages that handle the communication with UQCLOUD, while providing a near-native UQLAB experience to the end user.

The first such software binding, called UQ[py]Lab, is developed for PYTHON and will be presented next. Note that, while the software bindings are indeed language-specific, they consist only of the few hundreds lines of code needed to prepare and interpret the JSON structured messages needed by the API, a much more manageable cost w.r.t. translating and maintaining the entire UQLAB software into multiple languages.

3.2 UQ[py]Lab: UQLAB in PYTHON

Arguably one of the more mainstream languages in data science, PYTHON was chosen as the first language for UQCLOUD bindings, resulting in the UQ[py]Lab package. Although UQ[py]Lab is still under development (beta stage at the time of writing), it already supports most of the UQLAB features[2].

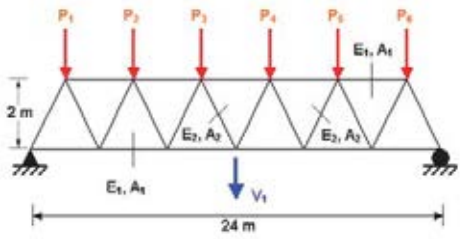
To showcase this tool, we revisit a textbook UQ problem that was included in [5], where the UQLAB framework was initially introduced. Consider the truss structure illustrated in Figure 4. It is characterised by 10 uncertain parameters, namely the Young moduli, E_1 and E_2, the beam sections A_1 and A_2 and the six variable loads, $P_1, \ldots, P_6$. The input variables are distributed according to independent lognormal or Gumbel distributions, parametrised by their first- and second-order moments and all variables are assumed statistically independent. The failure criterion (*limit state*) in this analysis corresponds to a threshold on the deflection at mid-span V_1, which is calculated by an in-house simple finite elements model (FEM), available locally in the native PYTHON file `truss.py` (within a function called `model`).

The code that runs the reliability analysis is given in Figure 4 in two flavours. On the left side, we showcase the standard UQLAB code within MATLAB, whereas on the right side we show the equivalent UQ[py]Lab code in PYTHON. In both cases, the sequence of commands is summarised as follows:

- An INPUT object is created, based on the provided information about the distributions of the input parameters.
- A MODEL object is created by specifying that it is based on a function in a file named `truss_model.m` (resp. `truss.py`, function `model`) in MATLAB (resp. PYTHON). In both cases, this script executes the same FEM code.

[2] `https://www.uqlab.com/features`

Truss structure:



Parameter distributions:

Name	Type	μ	σ/μ
E_1,E_2 (Pa)	Lognormal	2.1×10^{11}	10%
A_1 (m^2)	Lognormal	2.0×10^{-3}	10%
A_1 (m^2)	Lognormal	1.0×10^{-3}	10%
$P_1, \ldots, P_6$ (N)	Gumbel	5.0×10^4	15%

UQLab (Matlab) code:

```
Input.Marginals(1).Name = 'E1';
Input.Marginals(1).Type = 'Lognormal'
    ;
Input.Marginals(1).Moments = [2.1e11,
     2.1e10];
Input.Marginals(2).Name = 'E2';
...

myInput = uq_createInput(Input)

Model.mFile = 'truss_model';



myModel = uq_createModel(Model);

Analysis.Type = 'Reliability';
Analysis.LimitState.Threshold = 0.13;
Analysis.Method = 'IS';
Analysis.MaxSamples = 1e4;



myAnalysis = uq_createAnalysis(
    Analysis)
```

UQ[py]Lab (Python) code:

```
Input = {
  'Marginals' : [
  {'Name' : 'E1',
  'Type' : 'Lognormal',
  'Moments': [2.1e11, 2.1e10]},
  {'Name' : 'E2',
  ...
  ]}
myInput = uq.createInput(Input)

Model = {
  'Type': 'Model',
  'ModelFun':'truss.model'
}
myModel = uq.createModel(ModelOpts)

Analysis = {
  'Type' : 'Reliability',
  'LimitState':
    { 'Threshold': 0.13 },
  'Method': 'IS',
  'MaxSamples': 1e4
}
myAnalysis = uq.createAnalysis(
    Analysis)
```

Figure 4: Reliability analysis of a truss structure: problem representation (top left), uncertain input parameters (top right) and UQLAB/UQ[py]Lab pseudo-code to perform the analysis in MATLAB (bottom left) and PYTHON (bottom right).

- An ANALYSIS object is created to perform the reliability analysis. In this example, a maximum admissible mid-span displacement of $0.13\,\mathrm{cm}$ is specified, together with the reliability estimation method of choice, importance sampling. The maximum allowed cost is set to 10^4. After executing the `uq_createAnalysis()` function in MATLAB (resp. `uq.createAnalysis()` in PYTHON), the results are stored in the workspace, inside the `myAnalysis` variable.

Next, we would like to highlight the similarity and essential equivalence between the two versions of the code. The overall simplicity and almost natural-language-based syntax of the original UQLAB is fully preserved in its python-based counterpart, UQ[Py]Lab. Of course, language specific choices need to be made, due to the different data structures available in different languages. As an example, UQLAB makes extensive use of the MATLAB native *structures*, which are not available in PYTHON, where they are substituted instead by the equivalent *dictionaries*.

Although the user experience with UQ[py]Lab is reminiscent of UQLAB, there are significant differences in the actions that take place in the background. In the UQLAB case, all operations take place locally on the machine of the user. In the UQ[py]Lab case, most of the UQ computations (creation and evaluation of INPUT, MODEL or ANALYSIS objects) take place in the cloud instead. The main exception to this is related to computational model evaluations (non surrogate-based), which are mostly executed locally. This applies to the truss example, where the FEM code is wrapped inside a PYTHON script that runs locally.

Despite the remote execution, the service latency is minimal (order of milliseconds), providing a user experience that is very close to standalone local software.

4 CONCLUSIONS AND OUTLOOK

In this work we presented a short review of the state of the UQLAB project as a whole, with a focus on its latest offspring, UQCLOUD.

To address the growing need of powerful, easy-to-learn UQ software in diverse fields of applied science and engineering, we introduce UQCLOUD, a programming language- and OS-agnostic version of UQLAB that runs on the cloud in a SaaS paradigm. We demonstrated how it can be used through minimalistic software bindings to nearly replicate the full UQLAB experience without the need of a local MATLAB installation, in PYTHON.

Shifting to a cloud-based software framework has the potential to change the way UQ is performed, as it can remove the computational burden from the client device. Among others, we plan on providing bindings in a number of different languages, including Julia, C++/C#, R, and many others. The development of dedicated online services/mobile applications based on the UQCLOUD platform, e.g. for sensitivity and reliability analysis, is also planned.

5 ACKNOWLEDGMENTS

This paper is a part of the project “Surrogate Modeling for Stochastic Simulators (SAMOS)” funded by the Swiss National Science Foundation (Grant #200021_175524), whose support is gratefully acknowledged.

REFERENCES

[1] Baudin, M., Dutfoy, A., Iooss, B. and Popelin, A.-L. 2015 , Open turns: An industrial software for uncertainty quantification in simulation, *arXiv preprint arXiv:1501.05242* .

[2] De Rocquigny, E., Devictor, N. and Tarantola, S., eds 2008 , *Uncertainty in industrial practice – A guide to quantitative uncertainty management*, John Wiley & Sons.

[3] Eldred, M. S., Giunta, A. A., van Bloemen Waanders, B. G., Wojtkiewicz, S. F., Hart, W. E. and Alleva, M. P. 2006 , DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis, Technical report, Citeseer.

[4] Feinberg, J. and Langtangen, H. P. 2015 , Chaospy: An open source tool for designing methods of uncertainty quantification, *Journal of Computational Science* **11**, 46–57.

[5] Marelli, S. and Sudret, B. 2014 , UQLab: A framework for uncertainty quantification in Matlab, *in* Vulnerability, Uncertainty, and Risk (Proc. 2nd Int. Conf. on Vulnerability, Risk Analysis and Management (ICVRAM2014), Liverpool, United Kingdom), American Society of Civil Engineers, pp. 2554–2563.

[6] Marelli, S., Wicaksono, D. and Sudret, B. 2019, The UQLab project: steps toward a global uncertainty quantification community, *Proc. 13th Int. Conf. on Applications of Stat. and Prob. in Civil Engineering (ICASP13), Seoul, South Korea*.

[7] Merkel, D. 2014 , Docker: lightweight linux containers for consistent development and deployment, *Linux journal* **2014**(239), 2.

[8] National Research Council 2012 , *Assessing the reliability of complex models: mathematical and statistical foundations of verification, validation, and uncertainty quantification*, National Academies Press.

[9] Olivier, A., Giovanis, D., Aakash, B., Chauhan, M., Vandanapu, L. and Shields, M. D. 2020 , UQpy: A general purpose python package and development environment for uncertainty quantification, *Journal of Computational Science* **47**, 101204.

[10] Patelli, E., Broggi, M., Angelis, M. d. and Beer, M. 2014 , Opencossan: An efficient open tool for dealing with epistemic and aleatory uncertainties, *in* Vulnerability, Uncertainty, and Risk: Quantification, Mitigation, and Management, pp. 2564–2573.

[11] Richardson, L., Amundsen, M. and Ruby, S. 2013 , *RESTful Web APIs*, O'Reilly Media, Inc.

[12] Sudret, B. 2007 , *Uncertainty propagation and sensitivity analysis in mechanical models – Contributions to structural reliability and stochastic spectral methods*, Université Blaise Pascal, Clermont-Ferrand, France. Habilitation à diriger des recherches.