UQTestFuns: A Python3 library of uncertainty quantification (UQ) test functions

Damar Wicaksono [1¶] **and Michael Hecht** [1]

1 Center for Advanced Systems Understanding (CASUS) - Helmholtz-Zentrum Dresden-Rossendorf (HZDR), Germany ¶ Corresponding author

DOI: 10.21105/joss.05671

Software

- Review
- Repository
- Archive

Editor: Nikoleta Glynatsi

Reviewers:

- @Himscipy
- @AnjaliSandip

Submitted: 07 July 2023
Published: 30 October 2023

Summary

Researchers are continuously developing novel methods and algorithms in the field of applied uncertainty quantification (UQ). During the development phase of a novel method or algorithm, researchers and developers often rely on test functions taken from the literature for validation purposes. Afterward, they employ these test functions as a common ground to compare the performance of the novel method against that of the state-of-the-art methods in terms of accuracy and efficiency measures.

`UQTestFuns` is an open-source Python3 library of test functions commonly used within the applied UQ community. Specifically, the package provides:

- an **implementation with minimal dependencies** (i.e., NumPy and SciPy) **and a common interface** of many test functions available in the UQ literature
- a **single entry point** collecting test functions *and* their probabilistic input specifications in a single Python package
- an **opportunity for an open-source contribution**, supporting the implementation of new test functions and posting reference results.

`UQTestFuns` aims to save the researchers' and developers' time from having to reimplement many of the commonly used test functions themselves.

Statement of need

The field of uncertainty quantification (UQ) in applied science and engineering has grown rapidly in recent years. Novel methods and algorithms for metamodeling (surrogate modeling), reliability, and sensitivity analysis are being continuously developed. While such methods are aimed at addressing real-world problems, often involving large-scale complex computer models—from nuclear (Wicaksono et al., 2016) to civil engineering (Castellon et al., 2023), from physics (Adelmann, 2019) to biomedicine (Eck et al., 2015)—researchers and developers may prefer to use the so-called UQ test functions for validation and benchmarking purposes.

UQ test functions are mathematical functions taken as black boxes; they take a set of input values and produce output values. In a typical UQ analysis, the input variables are considered *uncertain* and thus modeled probabilistically. The results of a UQ analysis, in general, depend not only on the computational model under consideration but also on the specification of the input variables. Consequently, a UQ test function consists of both the specification of the function as well as probabilistic distribution specification of the inputs.

UQ test functions are widely used in the community for several reasons:

- Test functions are *fast to evaluate*, at least *faster* than their real-world counterparts.

- There are many test functions *available in the literature* for various types of analyses.
- Although test functions are taken as black boxes, *their features are known*; this knowledge enables a thorough diagnosis of a UQ method.
- Test functions provide *a common ground* for comparing the performance of various UQ methods in solving the same class of problems.

Several efforts have been made to provide relevant UQ test functions to the community. For instance, researchers may refer to the following online resources to obtain UQ test functions (the list is by no means exhaustive):

- The Virtual Library of Simulation Experiments (VLSE) (Surjanovic & Bingham, 2013): This site is arguably the definitive repository for (but not exclusively) UQ test functions. It provides over a hundred test functions for numerous applications; each test function is described on a dedicated page that includes implementations in MATLAB and R.
- The Benchmark proposals of GdR (GdR MASCOT-NUM, 2008): The site provides a series of documents that contain test function specifications.
- The Benchmark page of UQWorld (UQWorld, 2019): This community site provides a selection of test functions for metamodeling, sensitivity analysis, and reliability analysis exercises along with their implementation in MATLAB.
- RPrepo—a reliability problems repository (Rózsás & Slobbe, 2019): This repository contains numerous reliability analysis test functions implemented in Python. It is not, however, a stand-alone Python package.

Using these online resources, one either needs to download each test function separately[1] or implement the functions following the provided formula (in the programming language of choice).

As an alternative way for obtaining test functions, UQ analysis packages are often shipped with a selection of test functions of their own, either for illustration, validation, or benchmarking purposes. Examples from the applied UQ community in the Python ecosystem are (the numbers are as of 2023-06-30; once again, the list is non-exhaustive):

- SALib (Herman & Usher, 2017; Iwanaga et al., 2022): Six test functions mainly for illustrating the package capabilities in the examples.
- PyApprox (Jakeman, 2019): 18 test functions, including some non-algebraic functions for benchmarking purposes.
- Surrogate Modeling Toolbox (SMT) (Bouhlel et al., 2019, 2023): 11 analytical and engineering problems for benchmarking purposes.
- OpenTURNS (Baudin et al., 2017): 37 test functions packaged separately as `otbenchmark` (Baudin et al., 2021; Fekhari et al., 2021) for benchmarking purposes.

These open-source packages already provide a wide variety of functions implemented in Python. Except for `otbenchmark`, the problem is that these functions are part of the respective package. To get access to the test functions belonging to a package, the whole analysis package must be installed first. Furthermore, test functions from a given package are often implemented in such a way that is tightly coupled with the package itself. To use or extend the test functions belonging to an analysis package, one may need to first learn some basic usage and specific terminologies of the package.

`UQTestFuns` aims to solve this problem by collecting UQ test functions into a single Python package with a few dependencies (i.e., NumPy (Harris et al., 2020) and SciPy (Virtanen et al., 2020)). The package enables researchers to conveniently access commonly used UQ test functions implemented in Python. Thanks to a common interface, researchers can use the available test functions and extend the package with new test functions with minimal overhead.

Regarding its aim, `UQTestFuns` is mostly comparable to the package `otbenchmark`. Both also acknowledge the particularity of UQ test functions that requires combining a test function and

[1]except for RPrepo, which allows for downloading the whole repository.

the corresponding probabilistic input specification. There are, however, some major differences:

- One of the otbenchmark's main aims is to provide the OpenTURNS development team with a tool for helping with the implementation of new algorithms. As such, it is built on top of and coupled to OpenTURNS. UQTestFuns, on the other hand, has fewer dependencies and is leaner in its implementations; it is more agnostic with respect to any particular UQ analysis package.
- UQTestFuns is more modest in its scope, that is, simply to provide a library of UQ test functions implemented in Python with a consistent interface and an online reference (similar to that of VLSE (Surjanovic & Bingham, 2013)), and not, as in the case of otbenchmark, an automated benchmark framework[2] (Fekhari et al., 2021).

Package overview

Consider a computational model that is represented as an M-dimensional black-box function:

$$\mathcal{M}: \mathbf{x} \in \mathcal{D}_{\mathbf{X}} \subseteq \mathbb{R}^M \mapsto y = \mathcal{M}(\mathbf{x}),$$

where $\mathcal{D}_{\mathbf{X}}$ and y denote the input domain and the quantity of interest (QoI), respectively.

In practice, the exact values of the input variables are not exactly known and may be considered uncertain. The ensuing analysis involving uncertain input variables can be formalized in the uncertainty quantification (UQ) framework following Sudret (2007) as illustrated in Figure 1.

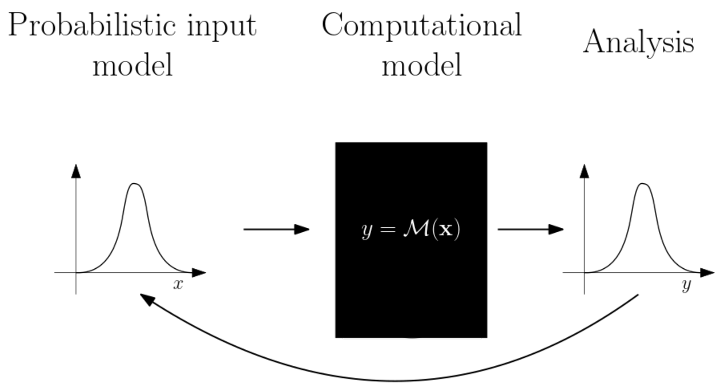

Figure 1: Uncertainty quantification (UQ) framework, adapted from Sudret (2007).

The framework starts from the center, with the computational model $\mathcal{M}$ taken as a black box as defined above. Then it moves on to the probabilistic modeling of the (uncertain) input variables. Under the probabilistic modeling, the uncertain input variables are replaced by a random vector equipped with a joint probability density function (PDF) $f_{\mathbf{X}}: \mathbf{x} \in \mathcal{D}_{\mathbf{X}} \subseteq \mathbb{R}^M \mapsto \mathbb{R}$.

Afterward, the uncertainties of the input variables are propagated through the computational model $\mathcal{M}$. The quantity of interest y now becomes a random variable:

$$Y = \mathcal{M}(\mathbf{X}),\ \mathbf{X} \sim f_{\mathbf{X}}.$$

This leads to various downstream analyses such as *reliability analysis*, *sensitivity analysis*, and *metamodeling* (or *surrogate modeling*). In UQTestFuns, these are currently the three main classifications of UQ test functions by their applications in the literature[3].

[2]A fully functional benchmark suite may, however, be in the future built on top of UQTestFuns.
[3]The classifications are not mutually exclusive; a given UQ test function may be applied in several contexts.

Reliability analysis

To illustrate the test functions included in `UQTestFuns`, consider the circular pipe crack reliability problem, a 2-dimensional function for testing reliability analysis methods (Li et al., 2018; Verma et al., 2015):

$$g(\mathbf{x}; \mathbf{p}) = \mathcal{M}(\mathbf{x}; t, R) - M = 4t\sigma_f R^2 \left(\cos\left(\frac{\theta}{2}\right) - \frac{1}{2} \sin\left(\theta\right) \right) - M,$$

where $\mathbf{x} = \{\sigma_f, \theta\}$ is the two-dimensional vector of input variables probabilistically defined further below; and $\mathbf{p} = \{t, R, M\}$ is the vector of (deterministic) parameters.

In a reliability analysis problem, a computational model $\mathcal{M}$ is often combined with another set of parameters (either uncertain or deterministic) to define the so-called *performance function* or *limit-state function* of a system denoted by g. The task for a reliability analysis method is to estimate the failure probability of the system defined as (Sudret, 2012):

$$P_f = \mathbb{P}[g(\mathbf{X}; \mathbf{p}) \leq 0] = \int_{\mathcal{D}_f = \{\mathbf{x} | g(\mathbf{x};\mathbf{p}) \leq 0\}} f_{\mathbf{X}}(\mathbf{x}) \; d\mathbf{x}, \tag{1}$$

where $g(\mathbf{x}; \mathbf{p}) \leq 0.0$ is defined as a *failure state*. The difficulty of evaluating the integral above stems from the fact that the integration domain D_f is defined implicitly.

The circular pipe crack problem can be created in `UQTestFuns` as follows:

```
>>> import uqtestfuns as uqtf
>>> circular_pipe = uqtf.CircularPipeCrack()
```

The resulting instance is *callable* and can be called with a set of valid input values. The probabilistic input model is an integral part of a UQ test function; indeed, according to Equation 1, the analysis results depend on it. Therefore, in `UQTestFuns`, the input model following the original specification is always attached to the instance of the test function:

```
>>> print(circular_pipe.prob_input)
Name         : CircularPipeCrack-Verma2015
Spatial Dim. : 2
Description  : Input model for the circular pipe crack problem from Verma...
Marginals    :

  No.   Name     Distribution      Parameters          Description
-----  -------  --------------  -----------------  --------------------
    1  sigma_f      normal      [301.079   14.78 ]   flow stress [MNm]
    2   theta       normal        [0.503 0.049]    half crack angle [-]

Copulas      : None
```

This probabilistic input model instance can be used to transform a set of values in a given domain (say, the unit hypercube $[0, 1]^M$) to the domain of the test function.

The limit-state surface (i.e., where $g(\mathbf{x}) = 0$) for the circular pipe crack problem is shown in Figure 2 (left plot). In the middle plot, 10^6 random sample points are overlaid; each point is classified whether it is in failure (red) or safe domain (blue). The histogram (right plot) shows the proportion of points that fall in the failure and safe domain.

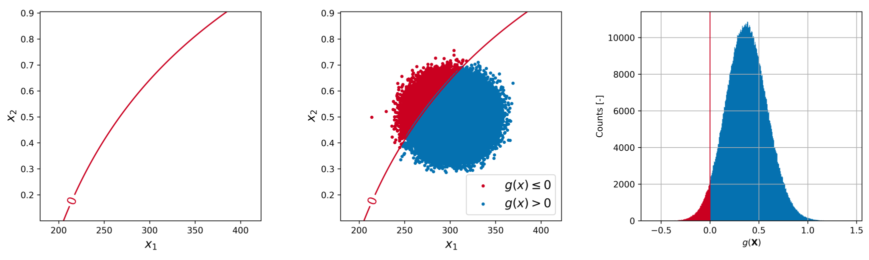

Figure 2: Illustration of reliability analysis: Circular pipe crack problem.

As illustrated in the previous series of plots, the task for a reliability analysis method is to estimate the probability where $g(\mathbf{X}) \leq 0$ as accurately and with as few model evaluations as possible. `UQTestFuns` includes test functions used in reliability analysis exercises in various dimensions having different complexities of the limit-state surface.

Sensitivity analysis

As another illustration, this time in the context of sensitivity analysis, consider the Sobol'-G function, an established sensitivity analysis test function (Saltelli & Sobol', 1995) included in `UQTestFuns`:

$$\mathcal{M}(\mathbf{x}) = \prod_{m=1}^{M} \frac{|4x_m - 2| + a_m}{1 + a_m},$$

where $\mathbf{x} = \{x_1, \ldots, x_M\}$ is the M-dimensional vector of independent uniform random variables in $[0, 1]^M$; and $\mathbf{a} = \{a_m = \frac{m-1}{2.0}, m = 1, \ldots, M\}$ is the set of (deterministic) parameters.

Unlike the previous test function example, the Sobol'-G test function is a variable-dimension test function and can be defined for any given dimension. For instance, to create a 6-dimensional Sobol'-G function:

```
>>> sobol_g = uqtf.SobolG(spatial_dimension=6)
```

As before, the probabilistic input model of the function as prescribed in the original specification is attached to the instance of the test function (i.e., the `prob_input` property).

The task of a sensitivity analysis method is to ascertain either qualitatively or quantitatively the most important input variables (for *factor prioritization*) or the least important input variables (for *factor fixing/screening*) with as few model evaluations as possible; for details on this topic, please refer to Saltelli et al. (2007) and Iooss & Lemaître (2015). `UQTestFuns` includes test functions used in sensitivity analysis exercises in various dimensions having different complexities in terms of the interactions between input variables.

Metamodeling

In practice, the computational model $\mathcal{M}$ is often complex. Because a UQ analysis typically involves evaluating $\mathcal{M}$ numerous times ($\sim 10^2$ — 10^6), the analysis may become intractable if $\mathcal{M}$ is expensive to evaluate. As a consequence, in many UQ analyses, a metamodel (surrogate model) is employed. Based on a limited number of model ($\mathcal{M}$) evaluations, such a metamodel should be able to capture the most important aspects of the input/output mapping but having much less cost per evaluation; it can, therefore, be used to replace $\mathcal{M}$ in the analysis.

While not a goal of UQ analyses per se, metamodeling is nowadays an indispensable component of the UQ framework (Sudret et al., 2017). `UQTestFuns` also includes test functions from the literature that are used as test functions in a metamodeling exercise.

Documentation

The online documentation of `UQTestFuns` is an important aspect of the project. It includes a detailed description of each of the available UQ test functions, their references, and when applicable, published results of a UQ analysis conducted using the test function. Guides on how to add additional test functions as well as to update the documentation are also available.

The package documentation is available on the `UQTestFuns` readthedocs.

Authors contribution statement and acknowledgments

The contributions to this paper are listed according to the CRediT (Contributor Roles Taxonomy). **Damar Wicaksono**: Conceptualization, methodology, software, validation, and writing - original draft. **Michael Hecht**: Conceptualization, writing - review and editing, project administration, and funding acquisition.

This work was partly funded by the Center for Advanced Systems Understanding (CASUS) that is financed by Germany's Federal Ministry of Education and Research (BMBF) and by the Saxony Ministry for Science, Culture and Tourism (SMWK) with tax funds on the basis of a budget approved by the Saxony State Parliament.

References

Adelmann, A. (2019). On nonintrusive uncertainty quantification and surrogate model construction in particle accelerator modeling. *SIAM/ASA Journal on Uncertainty Quantification*, *7*(2), 383–416. https://doi.org/10.1137/16m1061928

Baudin, M., Dutfoy, A., Iooss, B., & Popelin, A.-L. (2017). OpenTURNS: An industrial software for uncertainty quantification in simulation. In *Handbook of uncertainty quantification* (pp. 2001–2038). Springer International Publishing. https://doi.org/10.1007/978-3-319-12385-1_64

Baudin, M., Jebroun, Y., Fekhari, E., & Chabridon, V. (2021). otbenchmark. In *GitHub repository*. GitHub. https://github.com/mbaudin47/otbenchmark

Bouhlel, M. A., Hwang, J. T., Bartoli, N., Lafage, R., Morlier, J., & Martins, J. R. R. A. (2019). A Python surrogate modeling framework with derivatives. *Advances in Engineering Software*, *135*, 102662. https://doi.org/10.1016/j.advengsoft.2019.03.005

Bouhlel, M. A., Hwang, J., Bartoli, N., Lafage, R., Morlier, J., & Martins, J. (2023). Surrogate Modeling Toolbox. In *GitHub repository*. GitHub. https://github.com/SMTorg/smt

Castellon, D. F., Fenerci, A., Wiig Petersen, Øyvind, & Øiseth, O. (2023). Full long-term buffeting analysis of suspension bridges using Gaussian process surrogate modelling and importance sampling Monte Carlo simulations. *Reliability Engineering &Amp; System Safety*, *235*. https://doi.org/10.1016/j.ress.2023.109211

Eck, V. G., Donders, W. P., Sturdy, J., Feinberg, J., Delhaas, T., Hellevik, L. R., & Huberts, W. (2015). A guide to uncertainty quantification and sensitivity analysis for cardiovascular applications. *International Journal for Numerical Methods in Biomedical Engineering*, *32*(8). https://doi.org/10.1002/cnm.2755

Fekhari, E., Baudin, M., Chabridon, V., & Jebroun, Y. (2021). otbenchmark: An open source Python package for benchmarking and validating uncertainty quantification algorithms. *4th International Conference on Uncertainty Quantification in Computational Sciences and Engineering*. https://doi.org/10.7712/120221.8034.19093

GdR MASCOT-NUM. (2008). *Benchmark proposals of GdR*. Retrieved June 30, 2023, from https://www.gdr-mascotnum.fr/benchmarks.html.

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., & others. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Herman, J., & Usher, W. (2017). SALib: An open-source python library for sensitivity analysis. *The Journal of Open Source Software*, *2*(9), 97. https://doi.org/10.21105/joss.00097

Iooss, B., & Lemaître, P. (2015). A review on global sensitivity analysis methods. In *Uncertainty management in simulation-optimization of complex systems* (pp. 101–122). Springer US. https://doi.org/10.1007/978-1-4899-7547-8_5

Iwanaga, T., Usher, W., & Herman, J. (2022). Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses. *Socio-Environmental Systems Modelling*, *4*, 18155. https://doi.org/10.18174/sesmo.18155

Jakeman, J. D. (2019). PyApprox. In *GitHub repository*. GitHub. https://github.com/sandialabs/pyapprox

Li, X., Gong, C., Gu, L., Gao, W., Jing, Z., & Su, H. (2018). A sequential surrogate method for reliability analysis based on radial basis function. *Structural Safety*, *73*, 42–53. https://doi.org/10.1016/j.strusafe.2018.02.005

Rózsás, Á., & Slobbe, A. (2019). *Repository and black-box reliability challenge 2019*. Retrieved June 30, 2023, from https://gitlab.com/rozsasarpi/rprepo/.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., & Tarantola, S. (2007). *Global sensitivity analysis. The primer*. Wiley. https://doi.org/10.1002/9780470725184

Saltelli, A., & Sobol', I. M. (1995). About the use of rank transformation in sensitivity analysis of model output. *Reliability Engineering & System Safety*, *50*(3), 225–239. https://doi.org/10.1016/0951-8320(95)00099-2

Sudret, B. (2007). *Uncertainty propagation and sensitivity analysis in mechanical models —Contributions to structural reliability and stochastic spectral methods.* [Habilitation thesis]. Université Blaise Pascal - Clermont, France.

Sudret, B. (2012). Meta-models for structural reliability and uncertainty quantification. *Proceedings of the 5th Asian-Pacific Symposium on Structural Reliability and Its Applications*. https://doi.org/10.3850/978-981-07-2219-7_p321

Sudret, B., Marelli, S., & Wiart, J. (2017). Surrogate models for uncertainty quantification: An overview. *2017 11th European Conference on Antennas and Propagation (EUCAP)*. https://doi.org/10.23919/eucap.2017.7928679

Surjanovic, S., & Bingham, D. (2013). *Virtual library of simulation experiments: Test functions and datasets*. Retrieved June 30, 2023, from http://www.sfu.ca/~ssurjano.

UQWorld. (2019). *Benchmark page of UQWorld, the applied uncertainty quantification community*. Retrieved June 30, 2023, from https://uqworld.org/c/uq-with-uqlab/benchmarks.

Verma, A. K., Ajit, S., & Karanki, D. R. (2015). Structural reliability. In *Springer series in reliability engineering* (pp. 257–292). Springer London. https://doi.org/10.1007/978-1-4471-6269-8_8

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wicaksono, D., Zerkak, O., & Pautz, A. (2016). Global sensitivity analysis of transient code output applied to a reflood experiment model using the TRACE code. *Nuclear Science and Engineering*, *184*(3), 400–429. https://doi.org/10.13182/nse16-37