Available online at www.eccomasproceedia.org
Eccomas Proceedia UNCECOMP (2021) 218-231

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

OTBENCHMARK: AN OPEN SOURCE PYTHON PACKAGE FOR BENCHMARKING AND VALIDATING UNCERTAINTY QUANTIFICATION ALGORITHMS

Elias Fekhari[1], **Michaël Baudin**[1], **Vincent Chabridon**[1], **Youssef Jebroun**[1]

[1]EDF R&D, 6 Quai Watier, 78400 Chatou
e-mail: {elias.fekhari, michael.baudin, vincent.chabridon}@edf.fr

Abstract.

*Over the past decade, industrial companies and academic institutions pooled their efforts and knowledge to propose a generic uncertainty management methodology for computer simulation. This framework led to the collaborative development of an open source software dedicated to the treatment of uncertainties, called "OpenTURNS" (Open source Treatment of Uncertainty, Risk'N Statistics). This paper aims at presenting a new Python package, called "`otbenchmark`", offering tools to evaluate the performance of a large panel of uncertainty quantification algorithms. It provides benchmark classes containing problems with their reference values. Two categories of benchmark classes are currently available: reliability estimation problems (*i.e.*, estimating failure probabilities) and sensitivity analysis problems (*i.e.*, estimating sensitivity indices such as the Sobol' indices). This package can either be used for validating a new algorithm or automatically comparing various algorithms on a set of problems. Additionally, the package provides several convergence and accuracy metrics to compare the performance of each algorithm. To face high-dimensional problems, `otbenchmark` offers graphical tools to draw multidimensional events, functions and distributions based on cross-cuts visualizations. Finally, to ensure `otbenchmark`'s accuracy, a test-driven software development method has been adopted (using, among others, Git for collaborative development, unit tests and continuous integration). Ultimately, `otbenchmark` is an industrial platform gathering problems with reference values of their solutions and various tools to achieve a robust comparison of uncertainty management algorithms.*

Keywords: Benchmark, Uncertainty Quantification, Reliability Analysis, Sensitivity Analysis, Python, OpenTURNS.

ISSN:2623-3339 © 2021 The Authors. Published by Eccomas Proceedia.
Peer-review under responsibility of the organizing committee of UNCECOMP 2021.
doi: 10.7712/120221.8034.19093

1 INTRODUCTION

Complex computer simulation often requires implementing uncertainty management methods to evaluate associated risks, robustness and design margins. Several industrial companies and academic institutions pooled their efforts and knowledge to propose a generic uncertainty management methodology for computer simulation. This framework led to the collaborative development of an open source software dedicated to the treatment of uncertainties, called "OpenTURNS" (Open source Treatment of Uncertainty, Risk'N Statistics) [3]. Initially created by EDF R&D, Airbus Group and Phimeca Engineering, later joined by IMACS and ONERA, OpenTURNS is a generic, modular, transparent and multi-accessibility industrial software dedicated to serve several purposes (e.g., uncertainty quantification, uncertainty propagation, surrogate modeling, reliability, sensitivity analysis and calibration).

In the vein of a first bennchmark challenge organized in 2019 (the "Black-box Reliability Challenge" [12]), this paper aims at presenting a new Python module, called "`otbenchmark`"[1], which aims at providing several automatic tools to evaluate the performance of a large panel of uncertainty quantification algorithms by relying on the probabilistic programming framework offered by OpenTURNS. In other words, this module provides benchmark classes for OpenTURNS. It sets up a framework to create use-cases or problems associated with reference values. Such a benchmark problem may be used in order to check that a given algorithm works as expected and to measure its performance in terms of speed and accuracy.

Two categories of benchmark classes are currently provided: the first one is devoted to reliability estimation problems (i.e., estimating failure probabilities), the second one is devoted to sensitivity analysis problems (i.e., estimating sensitivity indices such as the Sobol' indices). `otbenchmark` is currently composed of 26 reliability problems and 4 sensitivity problems. For all these problems, reference solutions are provided. These solutions are obtained, either from a crude Monte Carlo estimation with a controlled convergence, or using (when possible) an exact resolution (e.g., provided by algebraic operations on input distributions). Additionally, `otbenchmark` provides several convergence and accuracy metrics to compare the performance of each algorithm. Finally, in order to perform a complete benchmark, a loop can be automatically set to evaluate a large panel of algorithms over the complete set of examples.

Graphical representations are often useful to help the analyst to understand the underlying behavior of complex reliability or sensitivity problems. Since many of these problems have dimensions larger than two, it raises numerous practical issues. `otbenchmark` offers graphical tools to draw multidimensional events, functions and distributions based on cross-cuts. Finally, to ensure `otbenchmark`'s accuracy, a test-driven software development method was followed (using, among others, Git and github.com for collaborative development, unit tests and continuous integration).

Thus, `otbenchmark` is an industrial benchmark platform for uncertainty management algorithms, and can be seen as a versatile tool offering diverse problems with corresponding solutions, robust metrics and graphical representations for high-dimensional problems.

In this paper, section 2 gives a formulation reminder for reliability and sensitivity problems, presents the probabilistic programming framework of OpenTURNS and demonstrates the added value of `otbenchmark` over the existing benchmark repositories. Section 3 defines the package's object-oriented architecture by detailing its main classes. Section 4 introduces the benchmark problems and the research for the most accurate associated reference values. Finally, section 5 is an illustrative example of the results automatically produced by the package

[1]Official repository: https://github.com/mbaudin47/otbenchmark

for a range of problems and algorithms.

2 MOTIVATIONS AND OBJECTIVES

2.1 Reliability analysis and sensitivity analysis formulations

Formally, the general scope of this paper is to address typical problems defined by considering a model given by $g : \mathbb{R}^d \rightarrow \mathbb{R}^p$, for which one considers a set of d uncertain inputs X_i ($i \in \{1, \dots, d\}$) gathered in a random vector $\boldsymbol{X} \in \mathbb{R}^d$. This random vector is associated to its joint probability distribution denoted in the following by the joint probability density function (PDF) $f_{\boldsymbol{X}}$, built from marginal densities and a copula. Propagating the uncertainties can be achieved through the following relationship:

$$Y = g(\boldsymbol{X}) \tag{1}$$

where the model output Y is a random variable (either univariate or multivariate). In the following, one will only assume a scalar output variable for the sake of clarity (note that OpenTURNS is not limited to scalar outputs since most of its classes are designed to naturally handle multivariate outputs). Based on this formulation, several types of analyses, associated to various *quantities of interests* can be solved. Among others, one will focus in this paper only on the following two types of analyses:

- Reliability analysis: in this case, one desires to compute a *failure probability*, denoted by p_{f} and defined through a *threshold event* E which characterizes the failure. This quantity of interest simply reads:

$$p_{\mathrm{f}} = \int_{\mathbb{R}^d} \mathbb{1}_E(\mathbf{x}) f_{\boldsymbol{X}}(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{2}$$

  Many standard or advanced algorithms can be used such as, typically, simulation-based ones (e.g., Monte Carlo sampling, importance sampling, subset sampling), approximation-based methods (FORM/SORM) or hybrid algorithms (e.g., FORM coupled with importance sampling, surrogate-model based strategies).

- Sensitivity analysis: in such as case, one desires to compute a *sensitivity index* (or a set of indices) which reflects the way an input (or a set of inputs) influence the variability of an output quantity of interest. For instance, by considering the variance of Y, one can compute the well-known Sobol' indices [14] as follows:

$$S_i = \frac{\mathrm{Var}\left[\mathbb{E}[Y|X_i]\right]}{\mathrm{Var}[Y]}, \qquad S_{T_i} = \frac{\mathbb{E}\left[\mathrm{Var}[Y|\boldsymbol{X}_{-i}]\right]}{\mathrm{Var}[Y]}, \tag{3}$$

  where S_i is the first-order index, S_{T_i} the total-order index of the variable X_i and $\boldsymbol{X}_{-i}$ stands for $\boldsymbol{X}$ without the i-th component. Several algorithms can be used to estimate these indices (e.g., sampling-based, surrogate-based or given-data algorithms) [8].

Other types of analyses (e.g., estimating the mean and variance of the model output, calibration, surrogate model fitting) will be further considered in future work.

These two types of analyses can be performed by using dedicated algorithms proposed in various open source or commercial software. OpenTURNS is one of them and provides several classes and algorithms to do so. In the next paragraph, one specifically focuses on a few core elements of the library and briefly explains why OpenTURNS is particularly well-dedicated to solve efficiently the problems presented hereabove.

2.2 OpenTURNS as a tool for uncertainty quantification

Originally founded by an industrial partnership between EDF R&D, Airbus and Phimeca Engineering in 2005, IMACS joined the OpenTURNS partnership in 2014, followed by ONERA in 2019. As an open source software developed under the LGPL license, OpenTURNS offers several features: if the core is written in C++, the application programming interface is in Python which makes it usable as a standard Python library. The architecture of the code is fully object-oriented and provide a large panel of classes and methods which are supported by a rigorous continuous integration process which provides a high-quality code for both industrial studies and research projects. The tool, in addition to its development and maintenance, is intensively used to support several industrial uncertainty quantification studies of the various partners.

OpenTURNS provides tools for what is usually called "probabilistic programming", a programming paradigm which facilitates the combination of probabilistic objects for statistical modeling. Among the OpenTURNS probabilistic object used in `otbenchmark`, the most iconic ones must be introduced for a better understanding.

The `Distribution` class defines the probability distribution function of a random variable. It provides more than one hundred methods, including `getMean`, `getStandardDeviation`, `computePDF`, `computeQuantile`, `drawPDF`, `getSample`, `computeConditionalCDF`, etc. Dealing with a set of random variables (a.k.a marginals) intertwined with a dependency model (e.g., variance-covariance matrix or copulas) a.k.a a random vector, can easily be modeled using OpenTURNS.

The `SymbolicFunction` is an efficient tool one can use when an analytical expression of the function is known. The additional advantage of this class is evaluating the gradient and hessian when mathematically defined.

The `ThresholdEvent` class defines the event which probability is to be estimated. It is based on a `RandomVector`, an operator and a threshold. The event occurs when the realization of the random vector exceeds the threshold.

Together with these fundamental tools, OpenTURNS contains several algorithms designed for efficiently solving a large variety of problems as the ones given in Eq. (2) and Eq. (3). However, such an environment would benefit from a dedicated benchmark platform to assess the performance of both the existing algorithms in the library together with any new proposed method which needs to be tested.

2.3 From current benchmark repositories to the `otbenchmark` package

According to Oxford English dictionary, a "benchmark" is "something that can be measured and used as a standard that other things can be compared with". As shown in this section, there is a long history of benchmark problems in various fields of applied mathematics (e.g., numerical methods, statistical software, optimization, design of computer experiments and uncertainty quantification).

Several fields in applied mathematics (e.g., linear algebra, numerical analysis) and computer simulation benefited from the development of test problems libraries (see, e.g., [17, 7, 6]). Uncertainty quantification naturally benefited from all the tools developed in those fields. However, uncertainty quantification problems have something specific that they can involve both of the previous fields (i.e., statistical inference, numerical integration, optimization). In the recent years, much effort has been put in the development of open source or commercial efficient uncertainty quantification platforms. However, benchmarking tools have not been deeply explored. Well-known repositories of benchmark problems are the Virtual Library of Simulation Experiments[2] managed by S. Surjanovic and D. Bingham [15]. [15], or the one maintained by the GDR Mascot-Num[3] research group. However, as a remark, one can mention that these two repositories mainly propose test functions in Matlab and R (not Python), but without aiming at providing an automated benchmark framework.

In 2018, concluding a workshop[4] organized by the Department of Structural Reliability at TNO (Netherlands Organization for Applied Scientific Research) about the computational challenges and aspects in the reliability analysis of engineering structures, several members of the community decided to create the first "Black-box Reliability Challenge", whose first edition happened in 2019 [12]. For this challenge, almost thirty benchmark problems have been provided to participants on a public repository[5]. More details about this challenge and the results can be found in the websites of the second workshop in 2020[6]. Thus, by providing a set of reliability problems in open source (available in their original form here[7]), this first initiative offered a great opportunity to start with the project of a dedicated benchmark tool: namely the proposed `otbenchmark` package.

2.4 Objectives of the proposed tool

The objective of the proposed package is twofold: firstly, the idea is to provide a benchmark tool for any potential external user (either an OpenTURNS user or anyone interested in performing a benchmark); secondly, to provide to the OpenTURNS development team a tool helping the implementation of new algorithms.

From the external user point of view, one can imagine mostly two possible scenarios:

- (Scenario #1) A user would like to design, implement and test a new algorithm which is not proposed in the library yet. Then, `otbenchmark` would provide a range of test problems and reference values so as to compare the performance of the new algorithm with respect to reference results (or other existing algorithms);
- (Scenario #2) A user would like to apply, all (or part of) the algorithms available in the library on a given user-defined problem (e.g., either an analytical or a real industrial black-box problem).

More generally, in the context of a real industrial problem (e.g., typically a reliability or sensitivity analysis of a complex, potentially costly-to-evaluate, simulation model), a user could be interested in using this benchmark module as a catalogue of test functions displaying various features (e.g., input dimension, independence/dependence of the inputs, distribution types, rareness of the failure probability) which could help him/her to test, choose and validate a choice of several "candidate algorithms".

In the next section, the core architecture of the `otbenchmark` package is presented.

[2]http://www.sfu.ca/ ssurjano/index.html
[3]https://www.gdr-mascotnum.fr/benchmarks.html
[4]https://www.reliabilitytno.com/
[5]https://rprepo.readthedocs.io/en/latest/index.html
[6]https://reliabilityworkshop2020.com/
[7]https://gitlab.com/rozsasarpi/rprepo/

3 ARCHITECTURE OF THE PACKAGE

3.1 Classes for reliability problems

In the sequel, `ot` denotes the short name for the `OpenTURNS` platform and associated classes. The basic class for reliability problems is the `ReliabilityBenchmarkProblem`, which defines a generic reliability problem. This class defines three constructor parameters:

- `name`: a string representing the name of the benchmark problem. This is a short string, typically less than a dozen of characters;
- `thresholdEvent`: a `ot.ThresholdEvent` object representing the event to estimate;
- `probability`: a `float` which represents the exact probability.

The essential information is the reference probability $p_{\mathrm{f,ref}}$, which should be as accurate as possible. The best possible accuracy for a Python `float` is 53 significant (binary) bits, which approximately corresponds to 15 (up to 17) decimal digits. If this accuracy is not available, then a reference value may be used, for example, obtained from a large Monte Carlo sample. In general, the exact probability should be a constant value, e.g., 0.123456789. However, we may be forced to compute this probability at the creation of the problem, for example if the threshold of the problem can be set at the creation of the object. In this case, the unit test must check that the default value of the parameters correspond to a reference constant value.

The `ReliabilityBenchmarkProblem` provides several methods, including `getEvent`, `getProbability` and `getName`, which returns the corresponding attributes. Moreover, other methods are provided, including pretty printing services. More importantly, the `computeBeta` method, which computes the Hasofer-Lind reliability index β_{HL} using the relationship $p_{\mathrm{f}} = \Phi(-\beta_{\mathrm{HL}})$ [10] based on the reference probability value.

In order to implement a specific reliability problem, a derived class is defined from the `ReliabilityBenchmarkProblem` mother class. For example, one can mention the `ReliabilityProblem54` derived from the `ReliabilityBenchmarkProblem` class to implement the so-called “RP54” problem. The practical implementation involves the definition of the input distribution of $\boldsymbol{X}$, the model function $g(\cdot)$ and the threshold value s. All these elements are defined within an instance of the `ThresholdEvent` class.

Specific reliability problems have specific attributes which can be provided in the constructor of the problem. For example, the `ReliabilityProblem28` class provides, as an optional extra, an attribute to set the mean and standard deviations of the input Gaussian distributions. The default values of these parameters are the ones which originally defines the reliability problem, but the user may want to modify these default values, for example to make the problem easier or more difficult or easier. Thus, the user can tune the problem regarding his specific needs, but has the drawback to require to update the reference probability $p_{\mathrm{f,ref}}$ depending on the actual parameters values: this cannot be done with guaranteed accuracy, but for a very limited number of problems.

The following piece of code provides a step-by-step illustration of how to create a problem, run an algorithm and compare the computed probability with a reference probability $p_{\mathrm{f,ref}}$. For the `RminusSReliability` (Resistance-Sollicitation) benchmark problem, the `ProbabilitySimulationAlgorithm` class from `OpenTURNS` is used to estimate the probability based on a sequential Monte Carlo algorithm. After running the algorithm, the probability is estimated with the `getProbabilityEstimate` method and the absolute error computed using the reference probability provided by the `getProbability` of the benchmark problem.

```python
# Import the packages
import openturns as ot
import otbenchmark as otb
# Select a problem, get the associated event and the reference probability
problem = otb.RminusSReliability()
event = problem.getEvent()
pfReference = problem.getProbability()
# Create a Monte Carlo algorithm and set its stopping criteria
algoProb = ot.ProbabilitySimulationAlgorithm(event)
algoProb.setMaximumOuterSampling(1000)
algoProb.setMaximumCoefficientOfVariation(0.01)
# Run the algorithm
algoProb.run()
# Get the result and compare it to the reference
resultAlgo = algoProb.getResult()
pf = resultAlgo.getProbabilityEstimate()
absoluteError = abs(pf - pfReference)
```

3.2 Classes for sensitivity analysis problems

The `SensitivityBenchmarkProblem` class defines a generic sensitivity analysis problem, depending on the following constructor parameters:

- `name`: a string representing the name of the benchmark problem. This is a short string, typically less than a dozen of characters;
- `distribution`: a `ot.Distribution` which represents the input distribution of the random vector $\boldsymbol{X}$;
- `function`: a `ot.Function` to define the model $g(\cdot)$;
- `firstOrderIndices`: a `ot.Point` representing the exact first-order Sobol' indices;
- `totalOrderIndices`: a `ot.Point` representing the exact total-order Sobol' indices.

The corresponding `get` methods provides a way to get the current values of the parameters, e.g., `getFirstOrderIndices` and `getTotalOrderIndices`.

In order to implement a specific sensitivity analysis problem, a derived class of the `SensitivityBenchmarkProblem` mother class is provided. This requires to know the reference first-order and total-order Sobol' indices for the problem of interest. For some sensitivity analysis problems, the user is given a leeway to customize more specific parameters. As an example, one can mention the G-Sobol' test function, for which the `GSobolSensitivity` class provides the optional parameter `a` which, by default, contains an array of three floating point numbers (equal respectively to 0, 9 and 99). In this case, the reference first- and total-order order Sobol' indices must be updated according to the actual values of these tuning parameters, which is easy for the G-Sobol' test function, but might be more difficult or impossible for some other problems.

3.3 Classes to manage the results of a benchmark

When a benchmark problem is run, it is interesting to compare the results obtained by various algorithms on this problem. To do so, one needs to define the set of features that one wants to assess.

The `ReliabilityBenchmarkResult` class is used to define the output of a reliability benchmark problem. Its constructor parameters are:

- `exactProbability`: a floating point number representing the exact probability;
- `computedProbability`: a floating point number representing the estimated probability;
- `numberOfFunctionEvaluations`: an integer representing the number of function evaluations.

Based on these parameters, the class computes several attributes:

- `absoluteError`: a floating point number representing the absolute error of the estimated probability;
- `numberOfCorrectDigits`: a floating point number representing the log-relative error in base 10;
- `numberOfDigitsPerEvaluation`: a floating point number representing the number of correct digits per function evaluation.

This last attribute measures the efficiency of the algorithm in computing the significant digits of the probability (typically, the larger the better).

3.4 Classes to perform a benchmark

The `ReliabilityBenchmarkProblemList` static method returns a list of benchmark problems available in the library. In the following example, one gets the list of problems and compute its length:

```python
benchmarkProblemList = otb.ReliabilityBenchmarkProblemList()
numberOfProblems = len(benchmarkProblemList)
```

The number of problems is currently equal to 26. Then the following script performs a loop over the problems and prints the name, the reference probability $p_{\mathrm{f,ref}}$ and the dimension of each problem:

```python
for i in range(numberOfProblems):
    problem = benchmarkProblemList[i]
    name = problem.getName()
    pfReference = problem.getProbability()
    event = problem.getEvent()
    antecedent = event.getAntecedent()
    distribution = antecedent.getDistribution()
    dimension = distribution.getDimension()
    print("#", i, ":", name, " : pfReference = ", pfReference, ",
          dimension=", dimension)
```

As a result, the previous script prints:

Distribution	Symbol	Parameters	PDF
Uniform	$\mathcal{U}$	(a, b)	$f_X(x) = \begin{cases} \frac{1}{b-a} & , x \in [a,b] \\ 0 & , x \notin [a,b] \end{cases}$
Normal	$\mathcal{N}$	(μ, σ)	$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Log-normal	$\mathcal{LN}$	(μ, σ)	$f_X(x) = \frac{1}{x} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right)$
Exponential	$\mathcal{E}$	λ	$f_X(x) = \lambda e^{-\lambda x}$
Gumbel	$\mathcal{G}$	(μ, β)	$f_X(x) = \frac{1}{\beta} \exp\left(-\frac{x-\mu}{\beta} - \exp\left(-\frac{x-\mu}{\beta}\right)\right)$

Table 1: Probability distribution parametrization used in `otbenchmark`.

```
# 0 : RP8  : pfReference =  0.000784 , dimension = 6
# 1 : RP14 : pfReference =  0.00752  , dimension = 5
# 2 : RP22 : pfReference =  0.00416  , dimension = 2
[...]
```

4 DESCRIPTION OF THE BENCHMARK PROBLEMS

4.1 Input random variable parametrization

The first element of a benchmark problem is the input random vector $\mathbf{X} = (X_1, \ldots, X_d) \in \mathbb{R}^d$. Although it might look obvious, precising the way random variables are defined is crucial in the context of a benchmark, especially since several distributions can be parametrized differently (e.g., the log-normal distribution). Table 1 provides the parametrization used in `otbenchmark`. All together, OpenTURNS provides 36 basic probability distributions, which allow the user to build an infinite number of distributions by truncating, transforming and combining them. Moreover, a large panel of copulas can be used to model input dependency.

4.2 Reliability problems description

As described in the previous sections, a reliability problem is defined by an input random vector $\boldsymbol{X}$, a function $g(\cdot)$, a threshold s and the corresponding reference failure probability $p_{\mathrm{f,ref}}$. One of the major challenge of a reliability benchmark is getting the most accurate reference failure probability. Without such an accurate reference solution, the benchmark is worthless. Depending on the complexity of the function, the distribution of the input random vector and the rareness of the failure event, computing $p_{\mathrm{f,ref}}$ can be more or less challenging. The safest way to achieve this regardless of the previous parameters is using the a very large crude Monte Carlo simulation, however, this method is very costly.

For some specific cases, one can compute an exact solution using quadrature methods. These techniques are extremely powerful since they allows to exactly compute the full output distribution Y without sampling the function. Overall, among the reference probabilities provided in `otbenchmark`, some are estimated with a huge Monte Carlo sampling, some are borrowed from [12] (and the associated references from the literature), while others are computed exactly using quadrature methods on the input distributions.

The implementation of the exact quadrature computation is problem-dependent and should progressively be applied to as many problems as possible. These reference values are continuously improved and offer interesting work perspectives. Table 2 presents the current reference values for $p_{\mathrm{f,ref}}$ available in `otbenchmark` and the corresponding values of the Hasofer-Lind reliability index β_{HL}. Note that the values tagged with an asterisk (*) differ from the initial reference probabilities provided by [12].

Table 2: Reliability problems definition.

Problem label	d	$p_{\mathrm{f,ref}}$	β_{HL}
RP8	6	7.84e-04	3.16
RP14	5	7.52e-03	2.42
RP22	2	4.16e-03	2.64
RP24	2	2.86e-03	2.76
RP25	2	6.14e-06	4.36
RP28	2	1.46e-07	5.11
RP31	2	1.8e-04	3.58
RP33	3	2.57e-03	2.8
RP35	2	3.54e-03	2.7
RP38	7	8.1e-03	2.48
RP53	2	3.13e-03	1.86
RP54	20	9.98e-04	3.09
RP55	2	5.6e-01*	-0.15
RP57	2	2.84e-02	1.91
RP60	5	4.56e-02	1.7
RP63	100	3.79e-04	3.36
RP75	2	1.07e-02	2.33
RP77	3	2.87e-07	5.0
RP89	2	5.43e-03	2.55
RP91	5	6.97e-04	3.19
RP107	10	2.92e-07	5.0
RP110	2	3.19e-05	4.0
RP111	2	7.65e-07	4.81
R-S	2	7.86e-02	1.41
Axial stressed beam	2	2.92e-02	1.89
Four-branch serial system	2	2.19e-03	2.85

4.3 Sensitivity problems description

The `otbenchmark` package currently contains 4 sensitivity analysis problems. More precisely, it includes the sum of several Gaussian random input variables, the product of several Gaussian random input variables, the G-Sobol' function [13] and the Ishigami function [9]. The first two test functions have variable dimensions. More sensitivity analysis problems will be added in the next release.

5 BENCHMARK RESULTS

This section presents how `otbenchmark` allows to compute and compare various metrics on multiple problems with several algorithms. At first, a simple "`for`" loop is performed to solve each problem with a list of algorithms (e.g., Monte Carlo, FORM, SORM, FORM-IS and Subset) and compute a list of metrics (e.g., failure probability, number of correct digits, absolute error). This loop uses the same structure as the one described in subsection 3.4 and various metrics can be directly computed using the methods described in the subsection 3.3.

In the following, a maximum simulation budget is set to $n_{\max} = 10^4$ calls. Such a value is set for illustration purposes here, even if this budget is clearly not enough to reach low failure probabilities proposed by some problems. The Monte Carlo algorithm stopping criterion used is this maximum number of function evaluations (i.e., $n_{\max}$). FORM and SORM methods are used with the Abdo-Rackwitz algorithm for the search of the design point [10, 1]. Regarding SORM, the Breitung approximation formula is used [4]. For the two approximation methods, the maximum number of function evaluations is set to $n_{\max}$, the maximum absolute error, maximum relative error, maximum residual error and maximum constraint error are set to 10^{-3}. The FORM-IS algorithm [10] first uses the FORM analysis to find the design point and then uses importance sampling with a Gaussian importance distribution in the standard space, centered on the design point. The FORM-IS and Subset algorithms [2] are used with the same stopping criterion as the Monte Carlo algorithm.

Overall, the results are in three dimensions and summarize both the problems, the algorithms and the metrics. To handle such a complex data structure, the multi-columns `DataFrame` class offered by the `pandas` package [16] is used. In addition to being a reference for data manipulation, `pandas` is known to well perform with multi-columns, provides powerful styling options and allows one to export any `DataFrame` to a LaTeX table. Table 3 is the result of an automated `pandas` export of the failure probabilities. It is obvious here that, with such a limited budget (i.e., $n_{\max}$), some problems are too difficult for some algorithms which fail to converge towards the reference failure probabilities. This explains the numerous zero values in Table 3 (or with an hyphen symbol when one does not want to run the Monte Carlo algorithm since one already knows it will not converge). Note that the purpose here is not to solve the problems but discuss the way `otbenchmark` works, produces, displays and compares the results.

This short illustration was performed on five algorithms but could easily be extended to many more reliability analysis algorithms available in OpenTURNS, including the adaptive directional stratification [11], the multiple design points strategy adapted to FORM/SORM [5] or FORM-system algorithm [10]. Adaptive surrogate-based algorithms will also be added in future work.

6 CONCLUSION

The `otbenchmark` package offers a open source benchmark tool for any user interested in performing reliability or sensitivity analysis, but more generally, to much more analyses usually encountered in uncertainty quantification. This versatile tool can serve many objectives such as helping in the development, testing and validation of new algorithms, or applying several algorithms to a given problem. It mainly relies on powerful classes and methods inherited from the OpenTURNS library, but also propose several new classes and dedicated tools specifically derived for benchmarking purposes. It gives access to a collection of problems with their reference solutions and allows to compare the performances of algorithms. Moreover, various tools to make this comparison more automated, robust and visual are available. Several convergence and accuracy metrics are provided to compare the algorithms performances, graphical tools and result tables are proposed to ease the results analysis. The quality of the reference values and, more generally, of the software are of paramount importance to ensure a consistent benchmark.

Table 3: Estimation of p_f for all the problems using 5 algorithms ($n_\mathrm{max} = 10^4$ calls).

	$p_\mathrm{f,ref}$	Monte Carlo	FORM	SORM	FORM-IS	Subset
RP8	7.840e-04	9.000e-04	6.599e-04	7.837e-04	7.737e-04	8.863e-04
RP14	7.520e-03	9.000e-04	7.003e-04	6.988e-04	7.598e-04	8.720e-04
RP22	4.160e-03	3.500e-03	6.210e-03	4.391e-03	4.259e-03	4.117e-03
RP24	2.860e-03	3.600e-03	6.209e-03	6.209e-03	2.749e-03	2.486e-03
RP25	6.140e-06	1.000e-04	2.105e-03	1.064e-05	4.644e-05	3.415e-05
RP28	1.460e-07	–	2.850e-08	0.000e+00	1.332e-07	1.756e-07
RP31	1.800e-04	2.300e-03	2.275e-02	2.275e-02	3.319e-03	3.919e-03
RP33	2.570e-03	1.600e-03	1.350e-03	1.350e-03	2.322e-03	2.718e-03
RP35	3.540e-03	3.000e-03	1.350e-03	2.134e-03	2.377e-03	3.430e-03
RP38	8.100e-03	8.500e-03	7.902e-03	8.029e-03	8.146e-03	7.848e-03
RP53	3.130e-02	3.260e-02	1.180e-01	2.986e-02	3.143e-02	2.971e-02
RP55	5.600e-01	5.660e-01	5.000e-01	1.093e-05	5.645e-01	5.655e-01
RP54	9.980e-04	1.100e-03	5.553e-02	3.552e-03	9.767e-04	9.611e-04
RP57	2.840e-02	2.950e-02	4.504e-01	0.000e+00	2.746e-02	2.772e-02
RP75	1.070e-02	1.030e-02	0.000e+00	0.000e+00	0.000e+00	9.409e-03
RP89	5.430e-03	5.000e-03	2.009e-09	2.009e-09	9.002e-05	5.460e-03
RP107	2.920e-07	–	2.867e-07	2.867e-07	2.896e-07	2.337e-07
RP110	3.190e-05	–	3.167e-05	3.167e-05	3.078e-05	7.116e-06
RP111	7.650e-07	–	0.000e+00	0.000e+00	0.000e+00	7.308e-07
RP63	3.790e-04	1.000e-04	1.000e+00	0.000e+00	0.000e+00	4.063e-04
RP91	6.970e-04	1.000e-03	6.984e-04	7.001e-04	6.964e-04	6.838e-04
RP60	4.560e-02	4.860e-02	4.484e-02	4.484e-02	4.503e-02	4.230e-02
RP77	2.870e-07	–	6.687e-02	6.687e-02	4.002e-07	3.683e-07
Four-branch serial system	2.186e-03	2.900e-03	0.000e+00	0.000e+00	0.000e+00	2.428e-03
R-S	7.865e-02	7.870e-02	7.865e-02	7.865e-02	7.792e-02	7.633e-02
Axial stressed beam	2.920e-02	2.690e-02	2.998e-02	2.933e-02	2.867e-02	2.936e-02

This is managed first by providing reference values as accurate as possible (which may involve, for example and when possible, exact quadrature calculations). Furthermore, this quality expectation is made achievable using software development methods which includes source version control (using Git), unit tests, continuous integration and collaborative development. As part of the `otbenchmark` development roadmap, one can mention the following prospects: some of the reference values will be updated using, as much as possible, exact quadrature methods to get the largest possible number of significant digits for reference values; sensitivity analysis will be extended to new problems and more investigated; other types of analyses will be also investigated (e.g., central tendency, calibration).

ACKNOWLEDGMENTS

The Authors are grateful to Régis Lebrun (Airbus) for his contributions to the package and for providing exact reference values for several benchmark problems and to Bertrand Iooss (EDF R&D) for his help in proofreading the paper.

REFERENCES

[1] T. Abdo and R. Rackwitz. *A New Beta-Point Algorithm for Large Time-Invariant and Time-Variant Reliability Problems. 3rd IFIP Working Conf.* 1990.

[2] S.-K. Au and J. L. Beck. “Estimation of small failure probabilities in high dimensions by subset simulation”. In: *Probabilistic engineering mechanics* 16.4 (2001), pp. 263–277.

[3] M. Baudin et al. “OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation”. In: *Handbook of Uncertainty Quantification*. Ed. by R. Ghanem, D. Higdon, and H. Owhadi. Cham: Springer International Publishing, 2017, pp. 2001–2038.

[4] K. Breitung. “Asymptotic approximations for multinormal integrals”. In: *Journal of Engineering Mechanics* 110.3 (1984), pp. 357–366.

[5] A. Der Kiureghian and T. Dakessian. “Multiple design points in first and second-order reliability”. In: *Structural Safety* 20.1 (1998), pp. 37–49.

[6] A. Genz. “Testing multidimensional integration routines”. In: *Proc. of international conference on Tools, methods and languages for scientific and engineering computation*. 1984, pp. 81–94.

[7] N. Higham. “Algorithm 694 A Collection of Test”. In: *ACM Transactions on Mathematical Software* 17.3 (1991).

[8] B. Iooss and P. Lemaître. “A Review on Global Sensitivity Analysis Methods”. In: *Uncertainty Management in Simulation-Optimization of Complex Systems: Algorithms and Applications*. Ed. by G. Dellino and C. Meloni. Boston, MA: Springer US, 2015. Chap. 5, pp. 101–122.

[9] T. Ishigami and T. Homma. “An importance quantification technique in uncertainty analysis for computer models”. In: *[1990] Proceedings. First International Symposium on Uncertainty Modeling and Analysis*. IEEE. 1990, pp. 398–403.

[10] M. Lemaire, A. Chateauneuf, and J.-C. Mitteau. *Structural Reliability*. ISTE Ltd. - Wiley, 2009. ISBN: 978-1-848-21082-0.

[11] M. Munoz Zuniga et al. “Adaptive directional stratification for controlled estimation of the probability of a rare event”. In: *Reliability Engineering & System Safety* 96.12 (2011), pp. 1691–1712.

[12] A. Rozsas and A. Slobbe. *Repository and Black-box Reliability Challenge 2019*. `https://gitlab.com/rozsasarpi/rprepo/`. 2019.

[13] A. Saltelli and I. M. Sobol'. "Sensitivity analysis for nonlinear mathematical models: numerical experience". In: *Matematicheskoe Modelirovanie* 7.11 (1995), pp. 16–28.

[14] I. M. Sobol. "Sensitivity estimates for nonlinear mathematical models". In: *Mathematical Modelling and Computational Experiments* 1 (1993), pp. 407–414.

[15] S. Surjanovic and D. Bingham. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. `http://www.sfu.ca/~ssurjano`. 2013.

[16] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: `10.5281/zenodo.3509134`. URL: `https://doi.org/10.5281/zenodo.3509134`.

[17] "The LINPACK 1000x1000 benchmark program." `http://www.netlib.org/benchmark/`.