OpenTURNS and its graphical interface

Michaël Baudin[1] Anne Dutfoy[1] Anthony Geay[1] Ovidiu Mircescu[1]
Thibault Delage[1] Aurélie Ladier[2] Julien Schueller[2]
Thierry Yalamas[2]

February 5, 2019

[1] EDF R&D. 6, quai Watier, 78401, Chatou Cedex - France, michael.baudin@edf.fr
[2] Phimeca Engineering. 18/20 boulevard de Reuilly, 75012 Paris - France, yalamas@phimeca.com

Abstract

OpenTURNS is an open source library for uncertainty propagation by probabilistic methods. Developed in a partnership of four industrial companies (EDF, Airbus, Phimeca and IMACS), it benefits from a strong practical feed-back. Classical algorithms of UQ are available : central dispersion, probability of exceedance, sensitivity analysis, metamodels and stochastic processes. Developped in C++, OpenTURNS is also available as a Python module and has gained maturity thanks to more than 10 years of development. However, there are situations where the engineer in charge of performing an uncertainty study does not want to use a programming language such as C++ or Python.

In this talk, we will present a basic tutorial of OpenTURNS in Python and will review the new features in the library, which include new incremental statistical estimators.

Contents

1 Introduction

The number of software for uncertainty quantification is already substantial, and this number is growing. OpenTURNS, for example, is a C++ library for uncertainty propagation by probabilistic methods. OpenTURNS is also available as a Python module and has gained maturity thanks to more than 10 years of development. However, there are situations where the engineer in charge of performing an uncertainty study does not want to use a programming language such as C++, Python (e.g. OpenTURNS) or Matlab. In this context, providing a graphical user interface (GUI) may allow to greatly increase the use of OpenTURNS and, more generally, of the UQ methodology.

2 OpenTurns

OpenTURNS[1, 4, 2] is an open source software, available as a C++ library and a Python interface. It works under the Linux and Windows environments. The key features of OpenTURNS are the following:

- open source initiative to secure the transparency of the approach,
- generic to the physical or industrial domains for treating of multi-physical problems,
- high performance computing,
- includes a variety of qualified algorithms in order to manage uncertainties in several situations,
- contains complete documentation (Reference Guide, Use Cases Guide, User manual, Examples Guide, and Developers' Guide).

OpenTURNS is available under the LGPL license.

The main features of OpenTURNS are uncertainty quantification, uncertainty propagation, sensitivity analysis and metamodeling.

Moreover generic wrappers allows to link OpenTURNS to any external code G.

OpenTURNS can be downloaded from its dedicated website www.openturns.org which offers different pre-compiled packages specific to several Windows and Linux environments. It is also possible to download the source files from the SourceForge server and to compile them within another environment: the OpenTURNS Developer's Guide provides advices to help compiling the source files.

3 A tutorial example : the flooding model

All along this paper, we illustrate our discussion with a simple application model that simulates the height of a river and compares it to the height of a dyke that protects industrial facilities (Figure 1). When the river height exceeds the one of the dyke, flooding occurs. This academic model is used as a pedagogical example in [5]. The model is based on a crude simplification of the 1D hydro-dynamical equations of SaintVenant under the assumptions of uniform and constant flowrate and large rectangular sections. It consists of an equation that involves the characteristics of the river stretch:

$$S = Z_v + H \quad \text{with} \quad H = \left(\frac{Q}{BK_s\sqrt{\frac{Z_m - Z_v}{L}}} \right)^{0.6}, \tag{1}$$

where S is the maximal annual overflow, H is the maximal annual height of the river, B is the river width and L is the length of the river stretch. In this paper, we set the values of L and B parameters :

$$L = 5000 \quad B = 300.$$

The other four input variables Q, K_s, Z_v and Z_m are defined in Table 1 with their probability distribution. The randomness of these variables is due to their spatio-temporal variability, our ignorance of their true value or some inaccuracies of their estimation. We make the hypothesis that the input variables are independent.

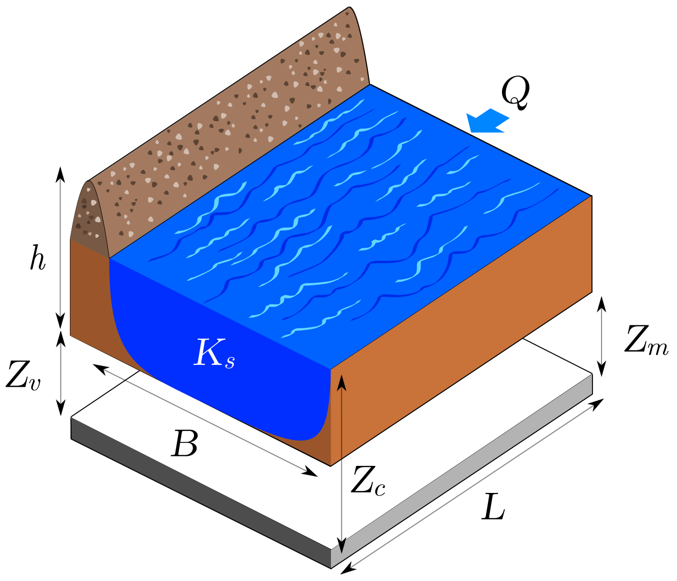

Figure 1: The flood example: simplified model of a river.

Input	Description	Unit	Probability distribution
Q	Maximal annual flowrate	m^3/s	Gumbel $\mathcal{G}(scale = 558, mode = 1013)$
K_s	Strickler coefficient	-	Normal $\mathcal{N}(30, 7.5)$
Z_v	River downstream level	m	Uniform $\mathcal{U}(49, 51)$
Z_m	River upstream level	m	Uniform $\mathcal{U}(54, 56)$

Table 1: Input variables of the flood model and their probability distributions.

The goal of this study is twofold:

- we want to estimate the mean river height $E(S)$,
- we want to perform the sensitivity analysis of the model, i.e. we want to rank the inputs Q, K_s, Z_v and Z_m.

4 Define the random vector

We beging by importing the required modules.

```
from openturns.viewer import View
import openturns as ot
from math import sqrt
import pylab as pl
```

We first define the function through which we want to propagate the uncertainties with the `def` operator.

```
def functionFlood(X) :
    Hd = 3.0
    Zb = 55.5
    L = 5.0e3
    B = 300.0
    Zd = Zb + Hd
    Q, Ks, Zv, Zm = X
    alpha = (Zm - Zv)/L
    H = (Q/(Ks*B*sqrt(alpha)))**(3.0/5.0)
    S = H + Zv
    return [S]
```

Then we convert this Python function into an OpenTURNS function with the `PythonFunction` class.

```
input_dimension = 4
g = ot.PythonFunction(input_dimension, 1, functionFlood)
```

Now we create the distributions for the input variables.

- There are several ways to set the parameters of the Gumbel distribution for the Q variable. Here the parameters are defined with the scale and mode parameters, which corresponds to the `GumbelAB` class.
- The Q and K_s variables must remain positive (a negative value is not compatible with the physical model). For this reason, we must truncate the distribution with `TruncatedDistribution`.

```
myParam = ot.GumbelAB(1013., 558.)
Q = ot.ParametrizedDistribution(myParam)
otLOW = ot.TruncatedDistribution.LOWER
Q = ot.TruncatedDistribution(Q, 0, otLOW)
Ks = ot.Normal(30.0, 7.5)
Ks = ot.TruncatedDistribution(Ks, 0, otLOW)
Zv = ot.Uniform(49.0, 51.0)
Zm = ot.Uniform(54.0, 56.0)
```

We set the descriptions of the random variables: they are used for the graphics.

```
Q.setDescription(["$Q␣(m^3/s)$"])
Ks.setDescription(["$Ks␣(m^{1/3})/s)$"])
Zv.setDescription(["Zv␣(m)"])
Zm.setDescription(["Zm␣(m)"])
```

The `drawPDF` presents the probability distribution function of the variable.

```
Q.drawPDF()
```

The following session produces the figure 2. When we closely look at the PDF of Q, we see a small increase of the density for $Q = 0$, because of the truncation of the distribution.

Then we create the input random vector `inputvector`: by default, the copula is independent. Finally, we create the output vector `S`.

```
X = ot.ComposedDistribution([Q, Ks, Zv, Zm])
inputRV = ot.RandomVector(X)
S = ot.RandomVector(g, inputRV)
```

These steps are typical of probabilistic programming: we have defined the random variables involved in the problem without having generating a sample *so far*.

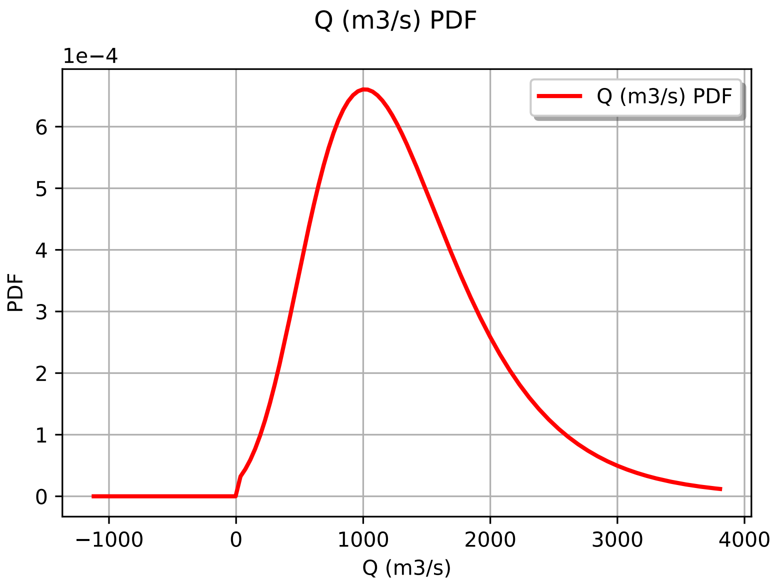

Figure 2: The probability density function of the variable Q.

5 Estimating the mean with an incremental algorithm

5.1 Theory

In this section, we the principles that are used in a new incremental algorithm in OpenTURNS 1.12 ; the goal of this algorithm is to estimate the mean of a random variable.

Assume that the output $Y \in \mathbb{R}^{n_Y}$ is a random vector and that we want to estimate the mean $E(Y)$.

The Monte Carlo algorithm with the sample mean :

$$\overline{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i$$

where n is the sample size and Y_i are i.i.d. realizations of the output.

We use the fact that the sample mean is asymptotically gaussian :

$$\overline{Y} \sim \mathcal{N}\left(E(Y), \frac{V(Y)}{n}\right).$$

where $V(Y)$ is the variance of the output and n is the sample size.

In general, most users set the sample size n in advance and estimate the precision afterwards. Instead, suppose that if set the absolute precision and wish to determine the smallest sample size n that achieves this precision. If we knew the variance $V(Y)$, we could set the value of n so that the standard deviation $\sqrt{V(Y)}$ would be small enough. If instead we want to set the relative precision, we could consider the coefficient of variation $\frac{\sqrt{V(Y)}}{E(Y)\sqrt{n}}$ as a criterion (if $E(Y) \neq 0$). However, we obviously do not know the value of neither $E(Y)$ nor $V(Y)$.

The purpose of the algorithm is to increase the sample size n incrementally until a stopping criteria is met. At each iteration, we approximate the values of $E(Y)$ and $V(Y)$ by their empirical estimators.

This algorithm works by updating a sample which size increases progressively until a stopping criteria is met. In order to get the best possible performance on distributed supercomputers and multi-core workstations, the size of the sample increases by block. For exemple, if the block size is equal to 100, then the sample size will be equal to 0, 100, 200, etc... On each block, the evaluation of the outputs can be parallelized, which allows to improve the performance of the algorithm.

There are three mathematical stopping criteria available:

- through an operator on the coefficient of variation $\frac{\sigma_i}{\mu_i}$ (a relative criterion)
- through an operator on the standard deviation σ_i (absolute criterion)
- on the maximum standard deviation per component: $\sigma_i \leq \max_{i=1,\ldots,n_Y} \sigma_i$ (absolute criterion)

By default, the maximum coefficient of variation is used, i.e. the algorithm stops when:

$$\max_{i=1,\ldots,n_Y} \frac{\sigma_i}{\mu_i} \leq max_{COV}.$$

5.2 Tutorial

In this section, we present how to use the `ExpectationSimulationAlgorithm` class in the tutorial flooding example. We set the maximum number of iterations with the `setMaximumOuterSampling` so that we use at most 1000 iterations. In order to evaluate the function with blocks of size 10, we use the `setBlockSize`. In this simulation, we use a relative stopping criteria and configure the maximum coefficient of variation to be equal to 0.001.

```
algo = ot.ExpectationSimulationAlgorithm(S)
algo.setMaximumOuterSampling(1000)
algo.setBlockSize(10)
algo.setMaximumCoefficientOfVariation(0.001)
```

The computationnaly intensive part of the simulation is associated with the `run` method.

```
algo.run()
```

Once the simulation is done, the `getResult` method allows to access to the results.

```
result = algo.getResult()
expectation = result.getExpectationEstimate()
print("Mean = %f " % expectation[0])
print("Number of calls to G = %d" % g.getCallsNumber())
```

The previous session prints the following output.

```
Mean = 52.520729
Number of calls to G = 500
Coef. of var.=0.000994
```

The estimate of the mean has a known asymptotical gaussian distribution, which can be retrieved with the `getExpectationDistribution` method.

```
expectationDistribution = result.getExpectationDistribution()
print(expectationDistribution)
View(expectationDistribution.drawPDF())
```

We emphasize that the output of the `getExpectationDistribution` method is a `Distribution` in the OpenTURNS sense: the whole information is available, not just a part of it, making the output as programmatically meaningful as possible.

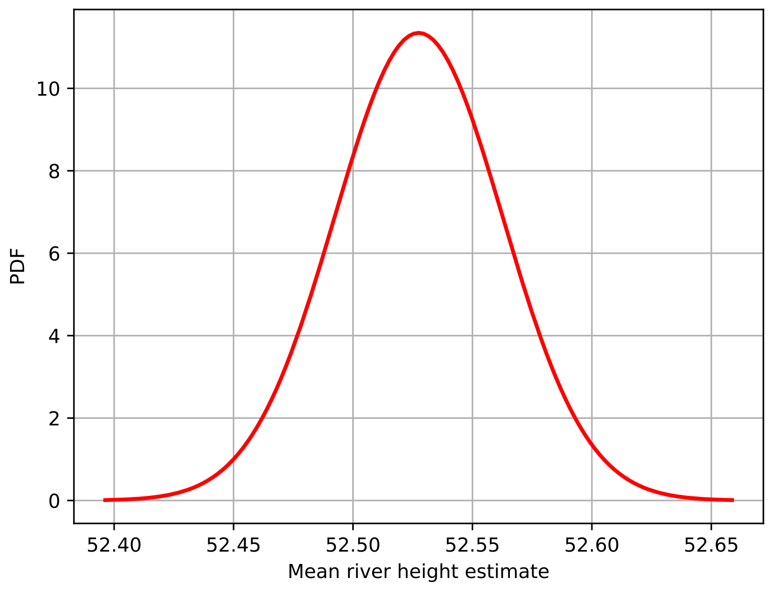

Figure 3: The probability density function of the estimate of the mean of the river height.

```
expectationDistribution = result.getExpectationDistribution()
expectationDistribution.drawPDF()
```

The previous script produces the figure 3. The figure shows that we have an accurate estimate of the mean, up to approximately 2 significant digits.

6 Estimate sensitivity indices with an incremental algorithm

6.1 Theory

In this section, we the principles that are used in a new incremental algorithm in OpenTURNS 1.12 which computes the Sobol' sensitivity indices.

In [6] the authors derive a new method to estimate the Sobol' sensitivity indices ; one of the advantages of the new estimator is that it is associated with an asymptotic distribution, which is derived thanks to the so called "delta"-method. Based on a suggestion by R.Lebrun, A. Dumas [3] used the same theoretical method in order to derive the asymptotic distribution of Sobol' sensitivity indices already available in OpenTURNS.

6.1.1 Overview

Let us denote by $X \in \mathbb{R}^{n_X}$ a random vector. This algorithm works in the general case where the output $Y \in \mathbb{R}^{n_Y}$ is a random vector : in this case it operates on aggregated indices. In order to simplify the discussion, let us make the hypothesis that $n_Y = 1$.

The Sobol' first order sensitivity indices are defined by

$$S_i = \frac{V(E(Y|X_i))}{V(Y)}$$

for $i = 1, ..., n_X$. The total order sensitivity indices are defined by :

$$T_i = 1 - \frac{V\left(E\left(Y|X_{-i}\right)\right)}{V(Y)}$$

where $-i$ is the set of indices which are different from i. In the remaining of this section, we focus on the first order sensitivity indice and let the reader consider [1] for the total order indices. Moreover, the derivation is the same for all input variables so that we omit the indice i in order to simplify the notations.

6.1.2 Asymptotic distribution

The algorithm is based on the fact that the estimatores of the first and total order Sobol' sensitivity indices asymptotically have the gaussian distribution. This gaussian distribution can be derived from the so called "delta"-method.

Assume that the Sobol' estimator is

$$\overline{S} = \Psi\left(\overline{U}\right)$$

where Ψ is a multivariate function, U is a multivariate sample and $\overline{U}$ is its sample mean. Each Sobol' estimator can be associated with a dedicated choice of function Ψ and vector U.

Let us denote by Φ_j^F (resp. Φ_j^T) the cumulated distribution function of the gaussian distribution of the first (resp. total) order sensitivity indice of the j-th input variable.

Each available estimator in the library provides its own distribution, namely the Saltelli, Mauntz-Kucherenko, Jansen and Martinez estimators.

6.1.3 Stopping criteria

We set $\alpha \in [0, 1]$ the level of the confidence interval and $\epsilon \in (0, 1]$ the length of the confidence interval. The algorithms stops when, on all components, one of the two following conditions are satisfied :

- first and total order indices haved been estimated with enough precision or
- the first order indices are separable from the total order indices.

The precision is said to be sufficient if the $1 - 2\alpha$ confidence interval is smaller than ϵ :

$$(\Phi_j^F)^{-1}(1-\alpha) - (\Phi_j^F)^{-1}(\alpha) \leq \epsilon$$

and

$$(\Phi_j^T)^{-1}(1-\alpha) - (\Phi_j^T)^{-1}(\alpha) \leq \epsilon$$

for $j = 1, ..., n_X$. The first order indices are *separable* from the total order indices if

$$\Phi_j^F(1-\alpha) \leq \Phi_j^T(\alpha)$$

for $j = 1, ..., n_X$. This criteria allows to stop when the algorithm has detected an interaction between input variables with sufficient precision.

6.2 Tutorial

In this section, we present how to use the `SaltelliSensitivityAlgorithm` classe in the tutorial flooding example.

We first set the parameters of the algoritms. We set the `alpha` variable so that a 90% confidence interval is used. In order to get confidence intervals which are not greater than 0.2, we set the variable `epsilon` variable accordingly. The block size corresponds to the size of the Sobol' design of experiment generated at each iteration. Finally, the `batchsize` variable contains the number of points evaluated simultaneously by the model.

```
alpha = 0.05 # 90% confidence interval
epsilon = 0.2 # Confidence interval length
blocksize = 50 # Size of Sobol experiment at each iteration
batchsize = 16 # Number of points evaluated simultaneously
```

Then we create the algorithm and configure the algorithm so that it uses the previous variables. Moreover, we configure the algorithm so that at most 100 iterations are used.

```
estimator = ot.SaltelliSensitivityAlgorithm()
estimator.setUseAsymptoticDistribution(True)
algo = ot.SobolSimulationAlgorithm(X, g, estimator)
algo.setMaximumOuterSampling(100) # number of iterations
algo.setBlockSize(blocksize)
algo.setBatchSize(batchsize)
algo.setIndexQuantileLevel(alpha) # alpha
algo.setIndexQuantileEpsilon(epsilon) # epsilon
algo.run()
```

Once that the algorithm has run, the results can be retrieved and estimates of first and total order indices can be printed.

```
result = algo.getResult()
fo = result.getFirstOrderIndicesEstimate()
to = result.getTotalOrderIndicesEstimate()
print("First order = %s" % (str(fo)))
print("Total order = %s" % (str(to)))
```

The previous script produces the following output.

```
First order = [0.529059,0.205321,0.381215,0.0316592]
Total order = [0.481355,0.130565,0.362292,0.011206]
```

We can obtain the asymptotic distribution of the first and total order indices. For example, the following script extracts the first component of the asymptotic distribution of the first order indice (which corresponds to the variable Q) and plots it.

```
dist_fo = result.getFirstOrderIndicesDistribution()
dist_fo_i = dist_fo.getMarginal(0)
graph = dist_fo_i.drawPDF()
graph.setTitle("S0")
graph.setXTitle("S0")
```

The previous script produces the figure 4.

In order to get a more compact view of the first and total order indices along with their confidence intervals, we often represent the 90% confidence intervals with a vertical bar. The figure 5 presents the Sobol' indices with asymptotic confidence intervals. We observe that the the confidence intervals are relatively small, as expected.

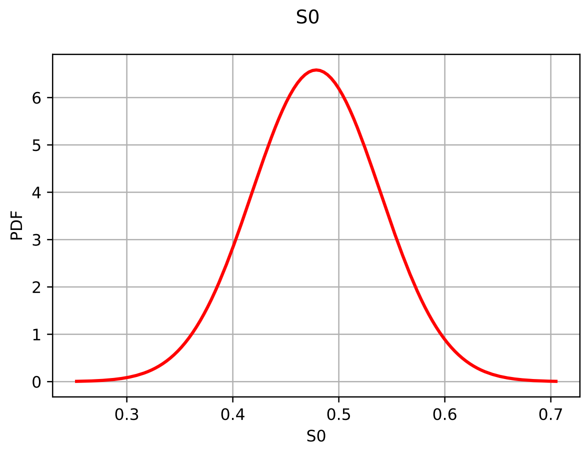

Figure 4: Asymptotic distribution of the first order Sobol' indices for the Q variable.

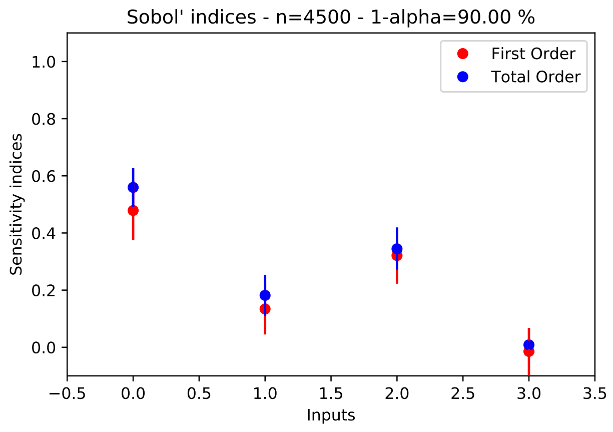

Figure 5: Sobol' indices with asymptotic confidence intervals.

References

[1] Michaël Baudin, Anne Dutfoy, Bertrand Iooss, and Anne-Laure Popelin. *Open-TURNS: An Industrial Software for Uncertainty Quantification in Simulation*, pages 1–38. Springer International Publishing, Cham, 2016.

[2] Michaël Baudin, Anne Dutfoy, Anthony Geay, Anne-Laure Popelin, Aurélie Ladier, Julien Schueller, and Thierry Yalamas. The graphical user interface of openturns, a uq software in simulation. UNCECOMP 2017, 15 - 17 June 2017 Rhodes Island, Greece. Eccomas Proceedia UNCECOMP (2017) 238-257 `https://www.eccomasproceedia.org/conferences/thematic-conferences/uncecomp-2017/5366`.

[3] Antoine Dumas. Lois asymptotiques des estimateurs des indices de sobol'. application de la méthode delta. EDF, by Phiméca Engineering.

[4] EADS, EDF, Phimeca Engineering, and IMACS. Openturns, an open source initiative for the treatment of uncertainties, risks'n statistics. `www.openturns.org`.

[5] B. Iooss and P. Lemaître. A review on global sensitivity analysis methods. In C. Meloni and G. Dellino, editors, *Uncertainty management in Simulation-Optimization of Complex Systems: Algorithms and Applications*. Springer, 2015.

[6] Alexandre Janon, Thierry Klein, Agnès Lagnoux, Maëlle Nodet, and Clémentine Prieur. Asymptotic normality and efficiency of two sobol index estimators. *ESAIM: Probability and Statistics*, 18:342–364, 2014.