UNCECOMP 2019
3rd ECCOMAS Thematic Conference on
Uncertainty Quantification in Computational Sciences and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou (eds.)
Crete, Greece, 24—26 June 2019

OPENTURNS AND ITS GRAPHICAL INTERFACE

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

[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

Keywords: Uncertainty Quantification.

Abstract. *OpenTURNS is an open source library for uncertainty propagation by probabilistic methods. Developed in a partnership of five industrial companies (EDF, Airbus, Phimeca, IMACS and ONERA), it benefits from a strong practical feedback. Classical algorithms of UQ are available : central dispersion, probability of exceedance, sensitivity analysis, metamodels and stochastic processes. Developed 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 context, providing a graphical user interface (GUI) may allow to greatly increase the use of OpenTURNS and, more generally, of the UQ methodology.

In this paper, we present a basic tutorial of OpenTURNS in Python and will review the new features in the library, which include new incremental statistical estimators. In the second part, we review the new features in the open source GUI will be presented.

1 Introduction

OpenTURNS 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 increase the use of OpenTURNS and, more generally, of the UQ methodology.

2 OpenTurns

OpenTURNS[**? ? ?**] 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 algorithms in order to manage uncertainties in several situations,
- contains complete documentation.

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 `www.openturns.org` which offers different precompiled packages specific to several Windows and Linux environments. It is also possible to download the source files from the Github server and to compile them within another environment: the OpenTURNS Developer's Guide provides advices to help compiling the source files. Finally, most Python users use `conda` or `pip` to install OpenTURNS.

3 A tutorial example : the flooding model

3.1 Introduction

In this paper, we illustrate our discussion with a simple application model that simulates the height of a river. The figure 1 presents the dyke that protects industrial facilities. When the river height exceeds the one of the dyke, flooding occurs. This academic model is used as a pedagogical example in [**?**]. The model is based on a crude simplification of the 1D hydrodynamical 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:

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

where Y 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

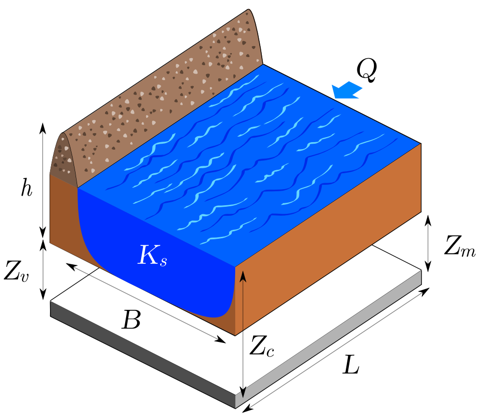


Figure 1: The flood example: simplified model of a river.

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.

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.

The goal of this study is twofold:

- we want to estimate the mean river height $E(Y)$,
- 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 with respect to their contributions to the variability of the output Y.

3.2 Define the random vector

In this section, we present the Python script which allows to define the output random vector in OpenTURNS.

We begin 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)
    Y = H + Zv
    return [Y]
```

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` method plots the the probability distribution function of the variable.

```
Q.drawPDF()
```

The previous 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 random vector `Y`.

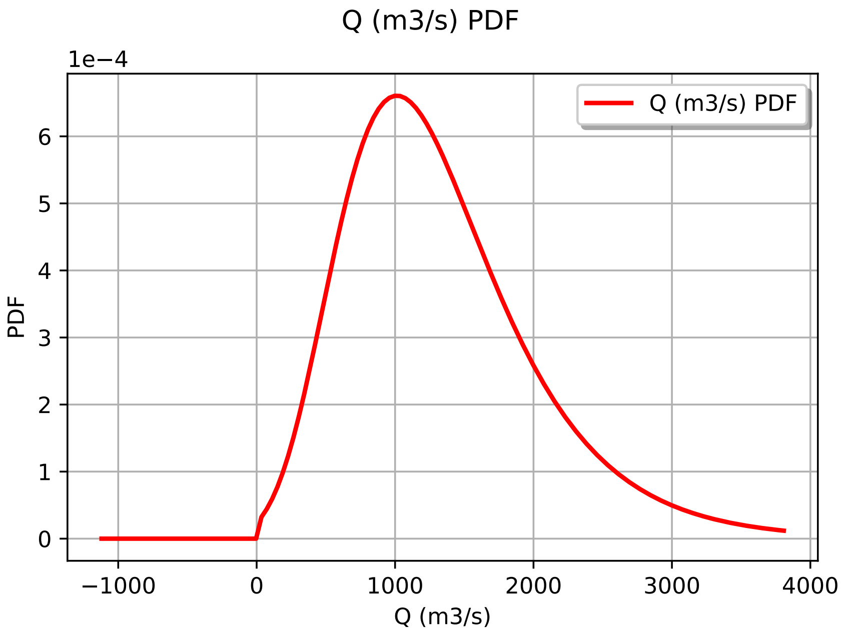


Figure 2: The probability density function of the variable Q.

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

4 Estimating the mean with an incremental algorithm

4.1 Theory

In this section, we present 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. Moreover, we would like to let the user as free as possible from the internal details of the algorithm and get the best possible performance on a supercomputer.

Assume that the output $Y \in \mathbb{R}^{n_Y}$ is a random vector and that we want to estimate the mean $E(Y_i)$ for $i = 1, ..., n_Y$.

The Monte Carlo method is based on the the sample mean:

$$\mu_i = \frac{1}{n} \sum_{j=1}^{n} y_i^{(j)}$$

for $i = 1, ..., n_Y$ where n is the sample size and $Y_j^{(i)}$ are i.i.d. outcomes of the random output.

The algorithm is based on the fact that the sample mean is asymptotically gaussian:

$$\mu_i \to \mathcal{N}\left(E(Y_i), \frac{V(Y_i)}{n} \right).$$

for $i = 1, ..., n_Y$ where $V(Y_i)$ is the variance of the i-th output and n is the sample size.

In general, most users set the sample size n in advance and estimate the precision afterwards. Let s_i be the (unbiased) sample standard deviation of the output Y_i:

$$s_i = \sqrt{\frac{1}{n-1}\sum_{j=1}^{N}\left(y_i^{(j)} - \mu_i\right)^2}$$

for $i = 1, ..., n_Y$. The absolute precision of the estimate μ_i can be evaluated based on the sample standard deviation of the estimator:

$$\sigma_i = \frac{s_i}{\sqrt{n}}$$

for $i = 1, ..., n_Y$. If $\mu_i \neq 0$ and $E(Y_i) \neq 0$, the relative precision can be estimated based on the coefficient of variation σ_i/μ_i for $i = 1, ..., n_Y$.

Instead, suppose that we set the absolute precision in advance and wish to determine the smallest sample size n that achieves this precision. If the variance $V(Y_i)$ is known (which rarely happens in practice), we can set the value of n so that the standard deviation $\sqrt{V(Y_i)}/\sqrt{n}$ is small enough. In the case where we want to set the relative precision, we can consider the coefficient of variation of the estimator $\frac{\sqrt{V(Y_i)}}{E(Y_i)\sqrt{n}}$ as a criterion (if $E(Y_i) \neq 0$). However, we generally do not know the values of neither $E(Y_i)$ nor $V(Y_i)$. This is why setting the sample size n in advance is not an easy task for the user in general.

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 $E(Y_i)$ and $V(Y_i)$ by their empirical estimators, which allows to evaluate the stopping criteria.

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 is equal to 100, 200, etc... On each block, the evaluation of the outputs can be parallelized, which allows to improve the performance of the algorithm. More details on this topic are presented in the section 6.4.

Since there are in general several outputs, i.e. $n_Y \geq 1$, we use a stopping criteria which is based on a operator. There are three mathematical stopping criteria available:

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

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

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

4.2 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` method. 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(Y)
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 the results.

```
result = algo.getResult()
expectation = result.getExpectationEstimate()
cv = result.getCoefficientOfVariation()[0]
print("Mean = %f " % expectation[0])
print("Number of calls to G = %d" % g.getCallsNumber())
print("Coef. of var.=%.6f" % (cv))
```

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. 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.

```
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.

5 Estimate sensitivity indices with an incremental algorithm

5.1 Theory

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

In [**?**] the authors derive a 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 [**?**] used the same theoretical method in order to derive the asymptotic distribution of Sobol' sensitivity indices already available in OpenTURNS.

5.1.1 Overview

Let us denote by $X \in \mathbb{R}^{n_X}$ the input random vector. Suppose that $Y = G(X) \in \mathbb{R}^{n_Y}$ is the corresponding output random vector, where G is the computer code. In this case the algorithm operates on aggregated indices. In order to simplify the discussion, let us make the hypothesis that there is only one output, i.e. $n_Y = 1$.

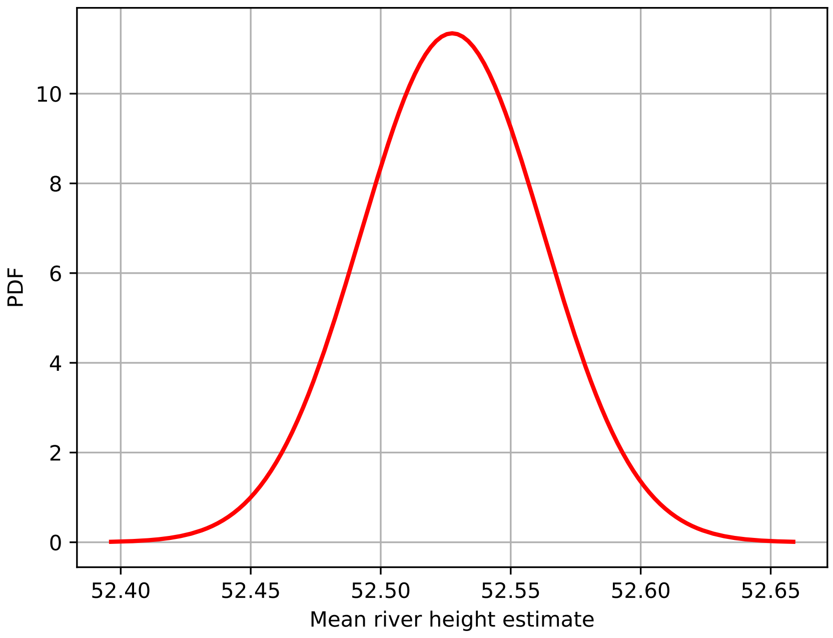


Figure 3: The probability density function of the estimate of the mean of the river height.

The Sobol' first order S_i and the total order sensitivity indices T_i are defined by

$$S_k = \frac{V(E(Y|X_i))}{V(Y)}, \qquad T_k = 1 - \frac{V\left(E\left(Y|X_{-i}\right)\right)}{V(Y)},$$

for $k = 1, ..., n_X$, 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 [**?**] 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.

5.1.2 Asymptotic distribution

The algorithm is based on the fact that the estimators 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.

Indeed, 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 specific choice of function Ψ and vector U. Therefore, the multivariate delta method implies:

$$\sqrt{n}\left(\overline{U} - \mu\right) \rightarrow \mathcal{N}\left(0, \nabla\psi(\mu)^T \Gamma \nabla\psi(\mu)\right)$$

where μ is the expected value of the Sobol' indice, $\nabla\psi(\mu)$ is the gradient of the function Ψ and Γ is the covariance matrix of $\overline{U}$. An implementation of the exact gradient $\nabla\psi(\mu)$ was derived for all estimators in OpenTURNS. In the algorithm, the unknown value μ is replaced by its estimator in order to compute the covariance matrix.

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

5.1.3 Stopping criteria

Let us denote by Φ_k^F (resp. Φ_k^T) the cumulated distribution function of the asymptotic gaussian distribution of the first (resp. total) order sensitivity indice of the k-th input variable, for $k = 1, ..., n_X$. 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 have 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 :

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

and

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

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

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

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

5.2 Tutorial

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

We first set the parameters of the algorithms. The `alpha` variable is set so that a 90% confidence interval is used. In order to get confidence intervals which are not greater than 0.1, 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 = 0.1 # 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 it so that it uses the previous variables. Moreover, we use the `setMaximumOuterSampling` method so that the algorithm uses at most 100 iterations.

```
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 = [0.575962,0.225763,0.357743,0.0216308]
Total order = [0.495489,0.176668,0.331708,0.00600383]
```

These estimates required 30 000 evaluations of the computer code.

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 confidence intervals are relatively small, as expected.

6 New features in the graphical user interface

6.1 Introduction

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 context, providing a graphical user interface (GUI) may allow to greatly increase the use of OpenTURNS and, more generally, of the UQ methodology.

This is why we develop since 2016 a graphical user interface (GUI) of OpenTURNS, which is integrated within SALOME [**?**]. This GUI is developed with the OpenSource LGPL license, which is the same as OpenTURNS and SALOME. SALOME binaries for the Linux platform are provided at the following URL:

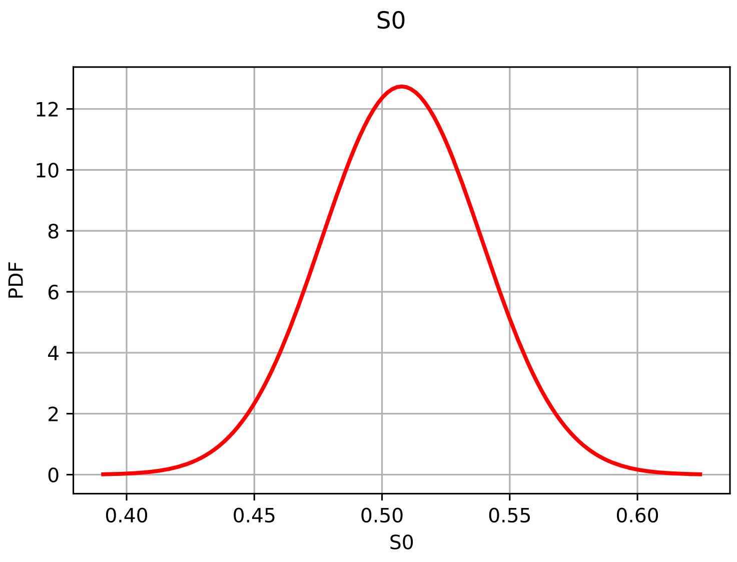


Figure 4: Asymptotic distribution of the first order Sobol' indices for the Q variable.

`https://www.salome-platform.org/contributions/edf_products`

The figure 6 presents the main window of the graphical user interface. The left pane contains the tree view which prints the opened studies and the main objects in each study. The right pane displays the main features of the interface, which makes the whole process easier for new users who might be unfamiliar with the uncertainty quantification methodology. The bottom pane is a Python console which allows to program the interface.

Each box in the right pane represents a single step in the global methodology ; the whole process is presented as a tree. Stop signs represent a method that cannot be used because a step must be fully completed before. When a new study is created, most boxes are greyed out, except the leftmost “Model definition” box, which require to define either a physical model (i.e. a computer code) or a data model (i.e. a CSV data file). Each time a step is completed, the corresponding steps which are then available are activated which allows the user to progress.

Details on the main features and the internal architecture of the GUI were already presented in [**?**], this is why this paper focuses on the new features.

6.2 Dependency structures

The GUI allows to define advanced dependency structures, based on copulas. The figure 7 presents the dialog box in which the copulas can be selected and configured.

The principle is to create sub-groups within the input variables. Within a given sub-group, we can select the copula and configure its parameters. Seven copulas are available: independent, Gaussian, Ali-Mikhail-Hak, Clayton, Farlie-Gumbel-Morgensten, Frank or an inference result.

For example, the figure 7 considers the situation in which the model has five inputs named X0, X1, X2, X3 and X4. In this particular model, the sub-group [X0,X1] is associated with the Gaussian copula while the sub-group [X3,X4] is associated with the Gumbel copula. The variable X2 remains independent from the others in this model.

Moreover, any multivariate sample can be used to estimate the parameters of a copula. In this case, the results of an inference can be reused in a dependency model.

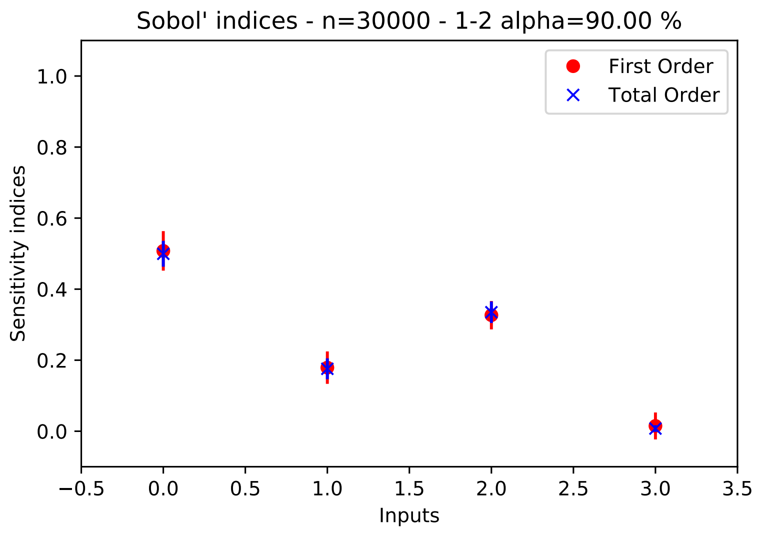


Figure 5: Sobol' indices with asymptotic confidence intervals.

6.3 Screening with the Morris method

The qualitative sensitivity analysis based on Morris's method [?] aims at selecting the significant input variables in a costly computer code which may have a large number of inputs. The GUI performs the screening analysis based on the OpenTURNS `otmorris` module [?].

The method makes use of a number of levels ℓ of levels, which is set by the user. Let $\Delta_i > 0$ be the step for the i-th input variable in the physical space, for $i = 1, ..., n_X$. This increment is computed from the number of levels L and the range of the i-th input variable.

The second parameter of the method is the number r of trajectories used in the design of experiment.

The k-th computed elementary effect associated to the i-th input marginal is the finite difference:

$$e_i^k = \frac{G\left(x_i^{(k')}\right) - G\left(x_i^{(k)}\right)}{\Delta_i}$$

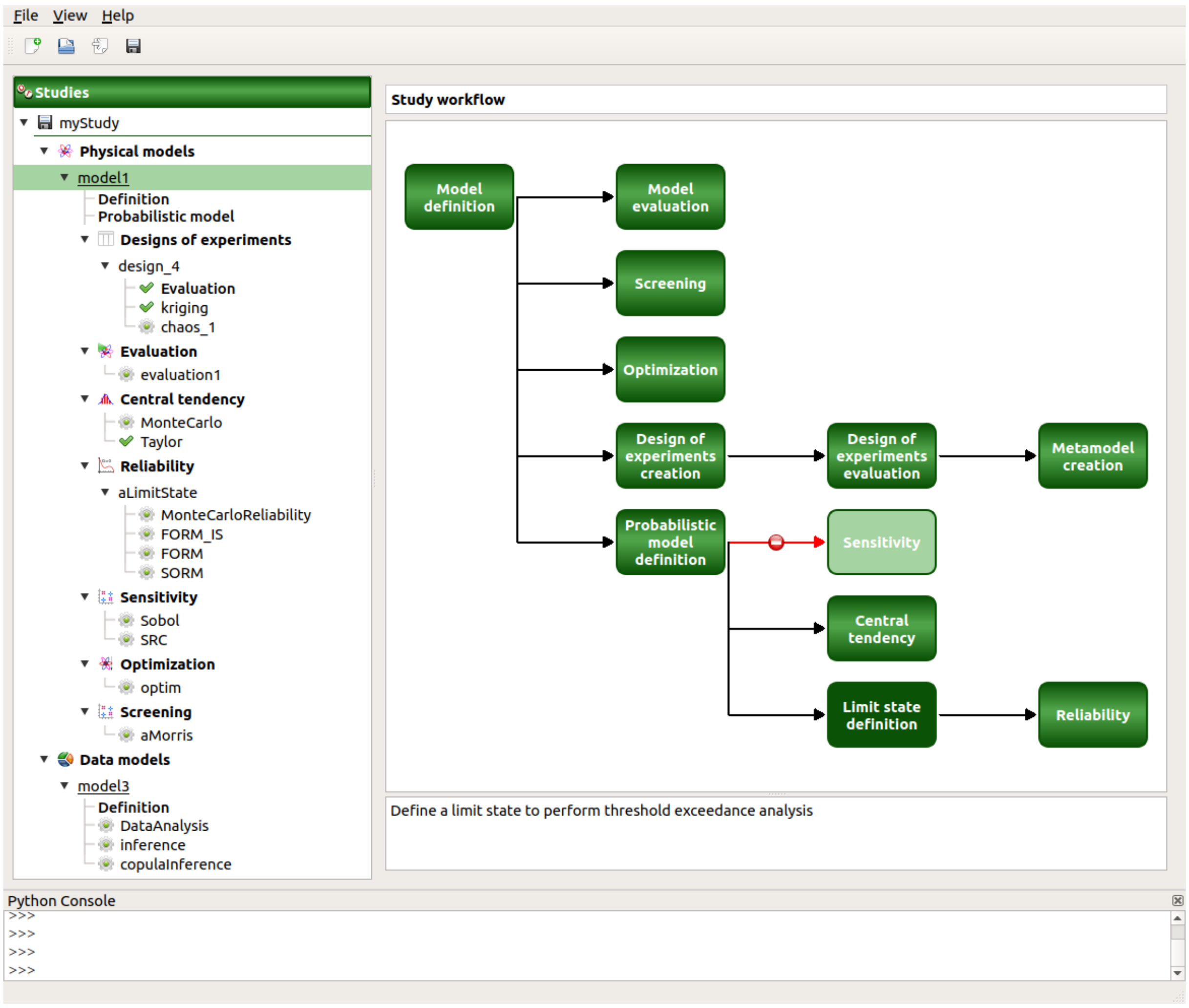
for $i = 1, ..., n_X$ and $k = 1, ..., r$. In the previous equation, the input points $x_i^{(k)}$ and $x_i^{(k')}$ are two points which differ from Δ_i in the physical input space. These points are computed based on a design of experiment which aims at grossly sampling the input space, generally with a rather large value of Δ_i.

The method computes μ_i, μ_i^* and σ_i, respectively the mean, absolute mean and the standard deviation of the elementary effects:

$$\mu_i = \frac{1}{r}\sum_{k=1}^{r} e_i^k, \qquad \mu_i^* = \frac{1}{r}\sum_{k=1}^{r} |e_i^k|, \qquad \sigma_i = \sqrt{\frac{1}{r}\sum_{k=1}^{r}(e_i^k - \mu_i)^2},$$

The goal of this method is to set the inputs variables into three classes, based on μ_i^* and the $\rho_i = \frac{\mu_i^*}{\sigma_i}$ factors:

1. if μ_i^* is close to zero, the i-th variable has no effect,



Define a limit state to perform threshold exceedance analysis

```
>>>
>>>
>>>
>>>
```

Figure 6: Main window of OpenTURNS' graphical user interface.

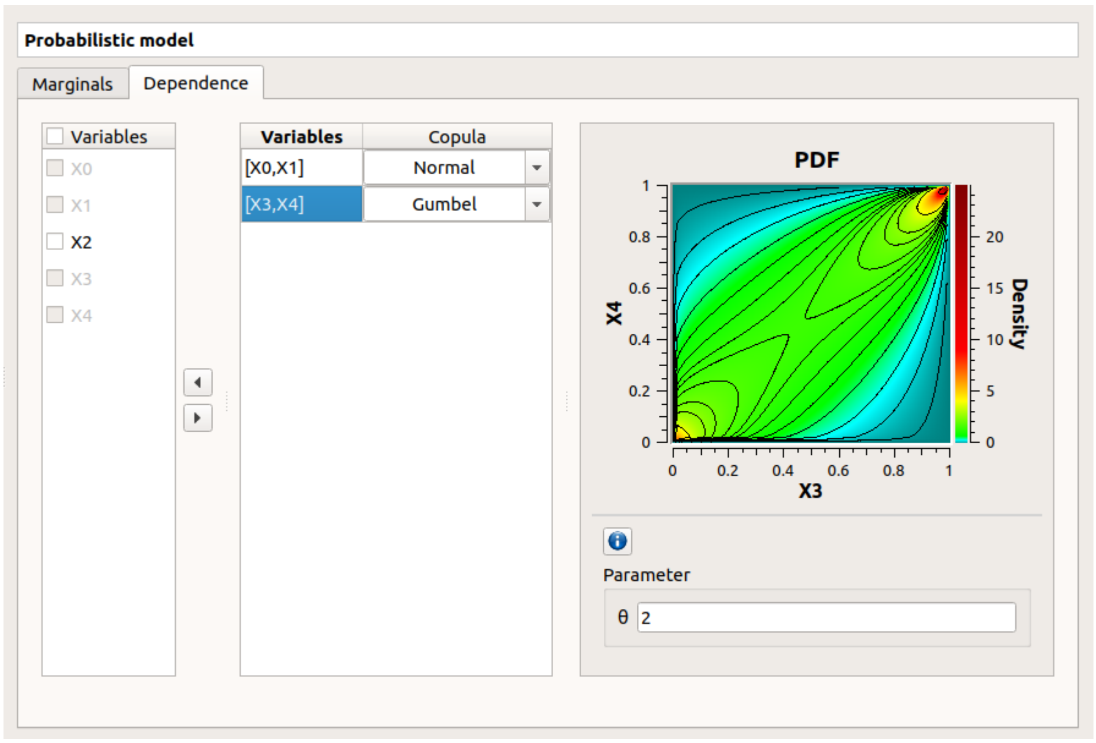


Figure 7: Managing a copula in the OpenTURNS GUI.

2. if $\rho_i \leq 0.5$ the i-th variable has almost linear effects,
3. if $\rho_i \geq 1$ the i-th variable has non-linear and non-monotonic effects

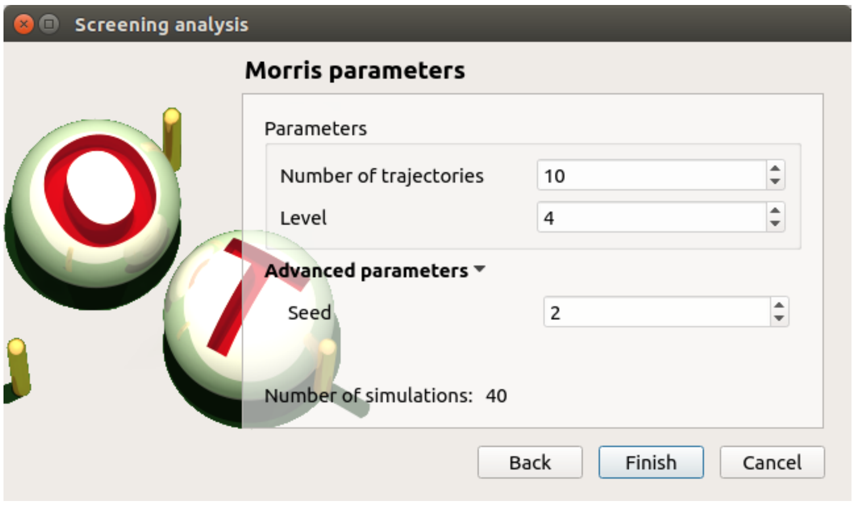
The figure 8 presents the dialog box which contains the parameters of the algorithm. The user can set the number of trajectories and the number of levels for each variable. The dialog box automatically computes the corresponding number of simulations and prints it in the bottom of the dialog box.

Once the simulations are performed, the figure 9 presents the results associated with a physical model which has 20 input variables. The main figure presents the mean and standard deviations of the elementary effects. A table (not shown in the figure) containing the list of input variables allows to see in which category fall each variable. A default classification is done by the GUI, but can be modified by the user.

6.4 Easy high performance computing

Within SALOME, users can access the remote high performance computing resources available at EDF R&D. Based on 16 100 cores, the Porthos supercomputer (2014) for example, can perform as high as 600Tflops (peak) [?]. The latest supercomputer at EDF R&D, Gaïa (2019), can perform as high as 3 052 Tflops (peak) [?] thanks to its 41 000 cores.

Within the GUI, the user can run simulations which are executed on a remote supercomputer with a minimum amount of configuration. The figure 10 presents the dialog box which is displayed in the context of a central tendency study based on a Monte-Carlo simulation. Enabling the *Parallelize status* checkbutton allows to select the computing resource available in the user's environment. The number of processes can be chosen by the user according to the hardware available and the amount of computing required by the simulation. Each job is associated with


Screening analysis

Morris parameters

Parameters

Number of trajectories: 10

Level: 4

Advanced parameters

Seed: 2

Number of simulations: 40

Figure 8: Performing screening with Morris's method in the GUI.

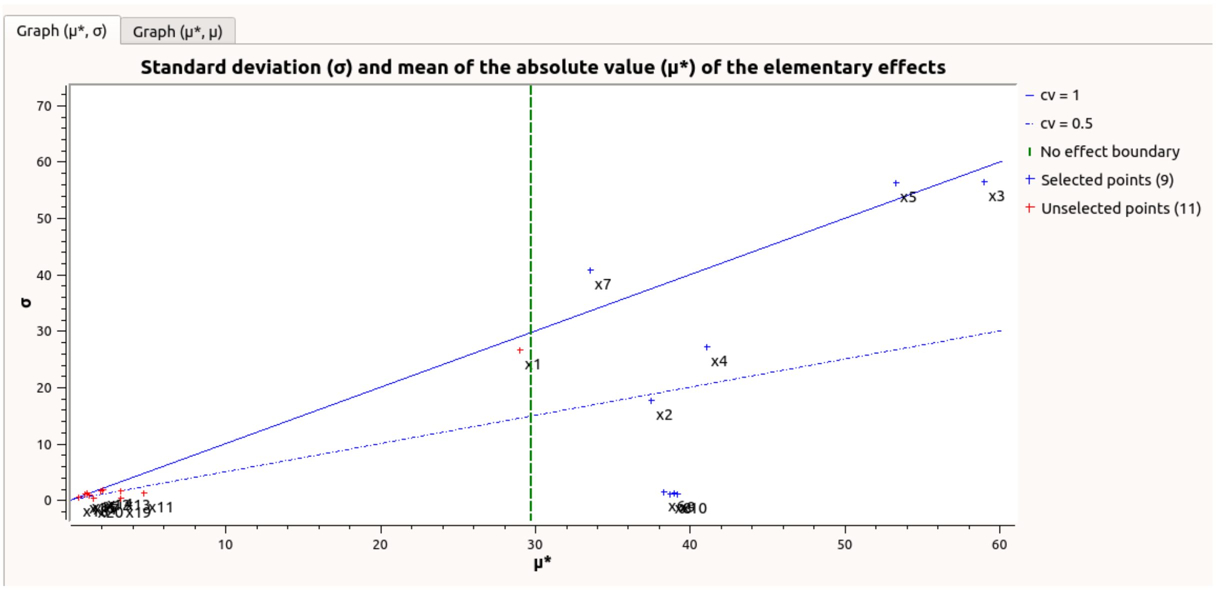


Figure 9: Results of the screening with Morris's method in the GUI.

a time limit which defines the maximum duration of one job. In most practical situations extra input files are used by the computer code (e.g. the mesh), which can be configured in the dialog box as well. The job submission is based on SLURM, but the user does not have to configure these low-level parameters which are handled automatically by the algorithms, with the principles which we now present.

The key point is to exploit the maximum possible amount of parallelism in the computations. However, between the start of the simulation and the end (which might take minutes, hours or days in the longest situations), most users want to regularily have a feedback on the execution of the simulation. For these reasons, the algorithms are performed based on blocks, which define a sub-sample on which the parallel computation can take place. At the end of each block, a progress bar is updated along with statistics which shows the number of executed simulations, the elapsed time and the value of the stopping criteria (e.g. the coefficient of variation of the mean estimator). A *Stop* button allows to interrupt the simulation.

Consider for example the situation presented in the figure 11, where a design of experiments involving 24 points must be evaluated. The parameters configured by the user in this example is the size of the block, which is set to 12, and the number of processors, which is set to 4. In this case, the simulation starts with a first job (e.g. a SLURM job) involving the points with indices from 1 to 12, and ends with a second job involving the points with indices 13 to 24. In both jobs, each processor is in charge of the evaluation of three points.

6.5 Perspectives: one-dimensional stochastic processes

In this section, we present the current developments of the GUI, which focuses on the management on stochastic fields.

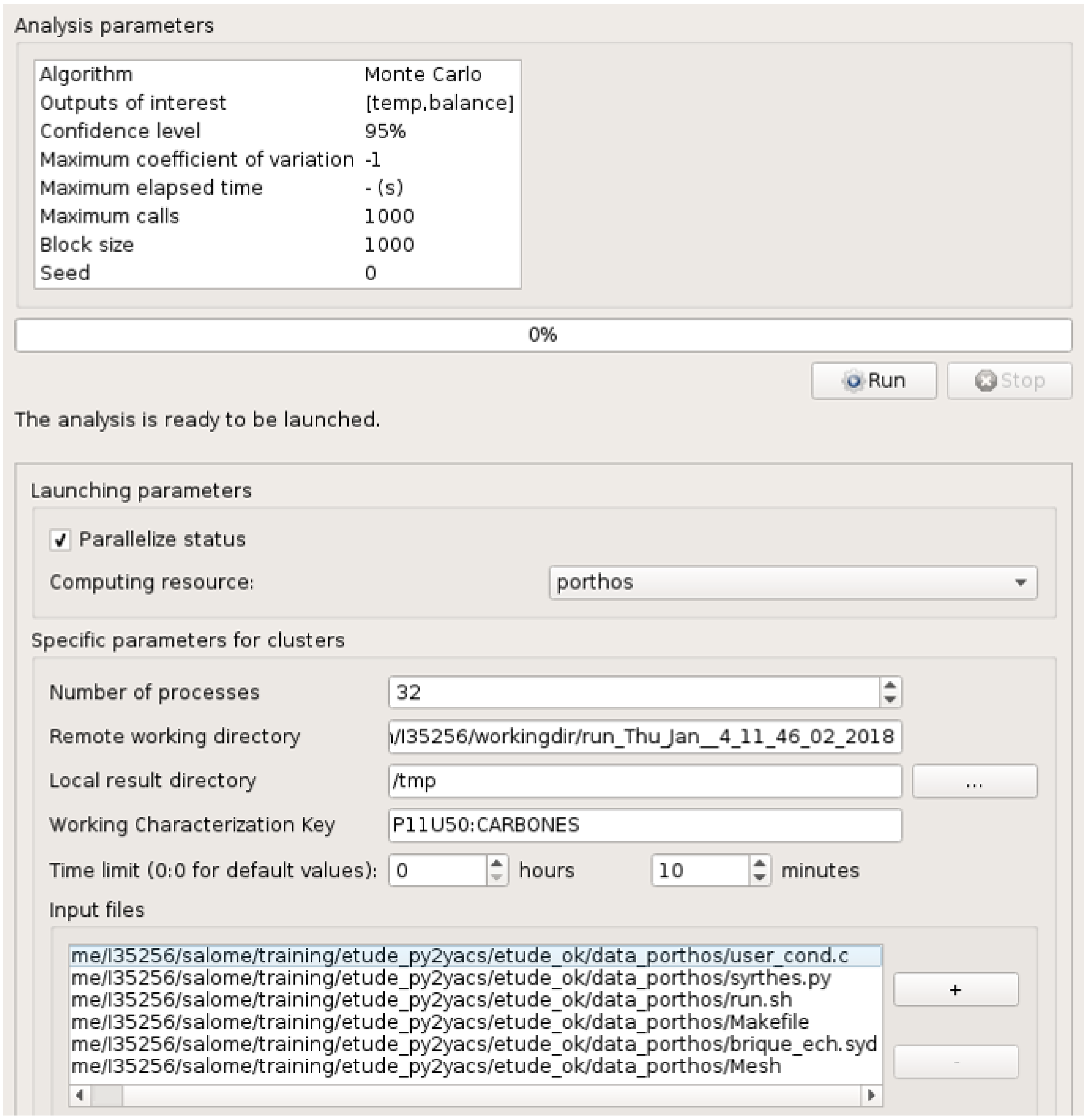
Indeed, there are various situations in which the simulator through which we propagate the uncertainties produces a stochastic process. This happens for example in the case where the simulator produces a time series or a one-dimensional spatial field.

The figure 12 presents a sample of trajectories in the GUI. In general, the sample size is large and this graphics does not convey much information, because the trajectories overlap and hide each other.

Obviously, this situation is more complex than the classical output random vector that many engineers are used to and require more advanced probabilistic methods. The most common way of managing such a situation is to use a dimension reduction method such as the functional principal component analysis or the Karhunen-Loève decomposition [**?**].

This is why Ribes et al [**?**] developed a new visualization tool in Paraview [**?**], based on a work by Kitware funded by EDF. This tool is the *functional bag chart*, also known as the highest density region plot in the bibliography [**?**]. The figure 13 presents the functional bag chart of a sample set of trajectories. This graphics allows to plot a functional boxplot in the sense that it plots a functional 95% confidence region. The graphics also allows to detect outlier trajectories, i.e. trajectories which achieve a low density in the reduced space.

The future version will extend these functional analyses to higher dimensions, including 2D stochastic fields.


Analysis parameters

Algorithm	Monte Carlo
Outputs of interest	[temp,balance]
Confidence level	95%
Maximum coefficient of variation	-1
Maximum elapsed time	- (s)
Maximum calls	1000
Block size	1000
Seed	0

0%

The analysis is ready to be launched.

Launching parameters

[✓] Parallelize status

Computing resource: porthos

Specific parameters for clusters

Number of processes	32
Remote working directory	\/I35256/workingdir/run_Thu_Jan__4_11_46_02_2018
Local result directory	/tmp
Working Characterization Key	P11U50:CARBONES
Time limit (0:0 for default values):	0 hours 10 minutes

Input files

```
me/I35256/salome/training/etude_py2yacs/etude_ok/data_porthos/user_cond.c
me/I35256/salome/training/etude_py2yacs/etude_ok/data_porthos/syrthes.py
me/I35256/salome/training/etude_py2yacs/etude_ok/data_porthos/run.sh
me/I35256/salome/training/etude_py2yacs/etude_ok/data_porthos/Makefile
me/I35256/salome/training/etude_py2yacs/etude_ok/data_porthos/brique_ech.syd
me/I35256/salome/training/etude_py2yacs/etude_ok/data_porthos/Mesh
```

Figure 10: Launching a parallel computation within the GUI.

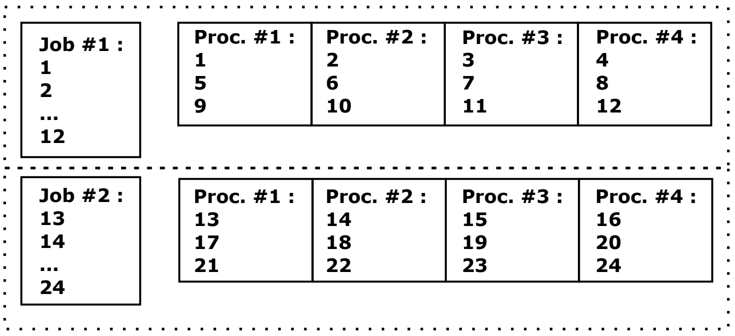


Figure 11: A block-based simulation performed on a supercomputer. We assume that we want to evaluate a design of experiments made of 24 points ; these points are numbered 1, 2, ..., 24. We consider a block size of 12 and a number of processors equal to 4. In this case, the block-based simulation uses two jobs.

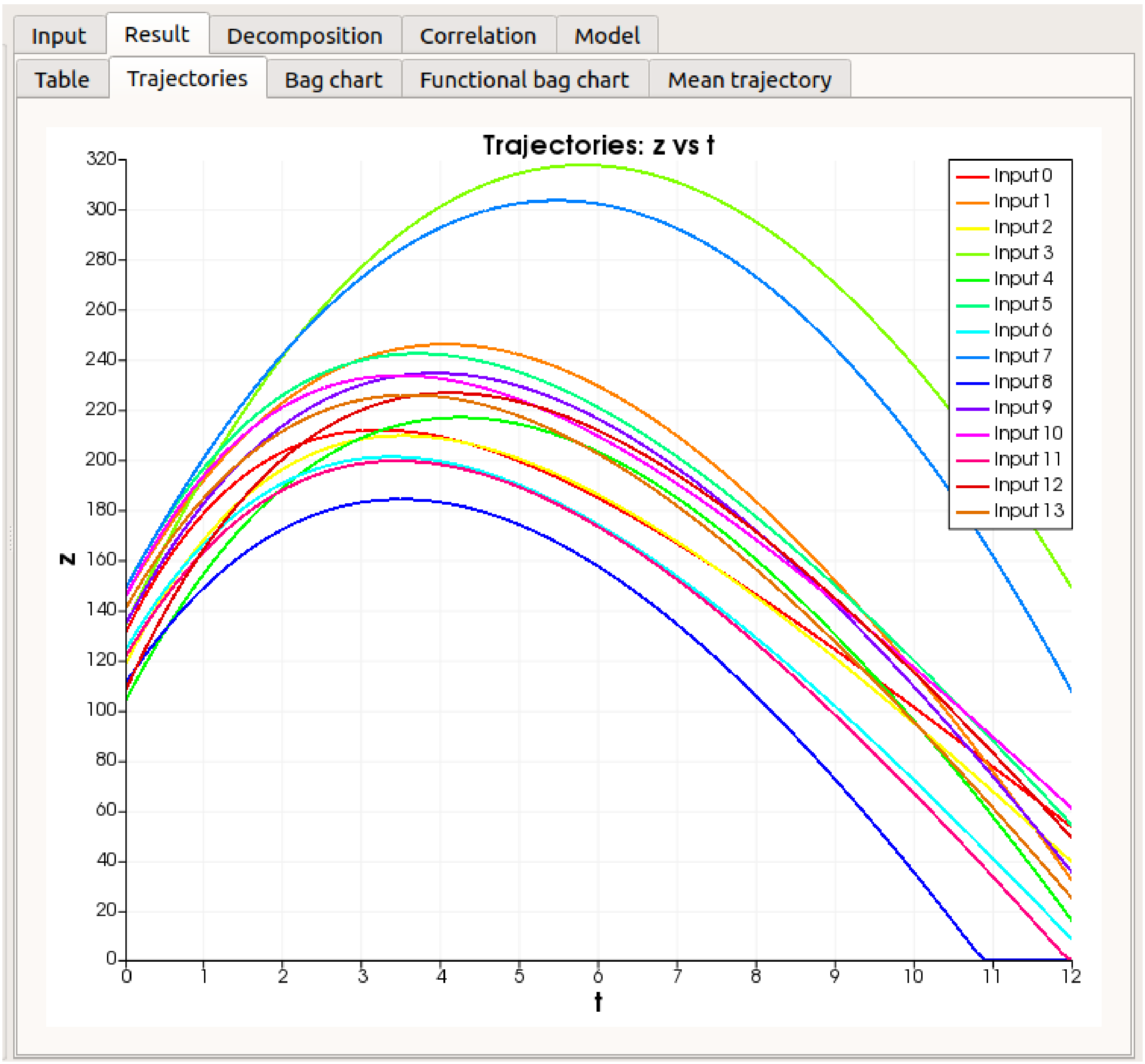


Figure 12: A sample of trajectories in the GUI.

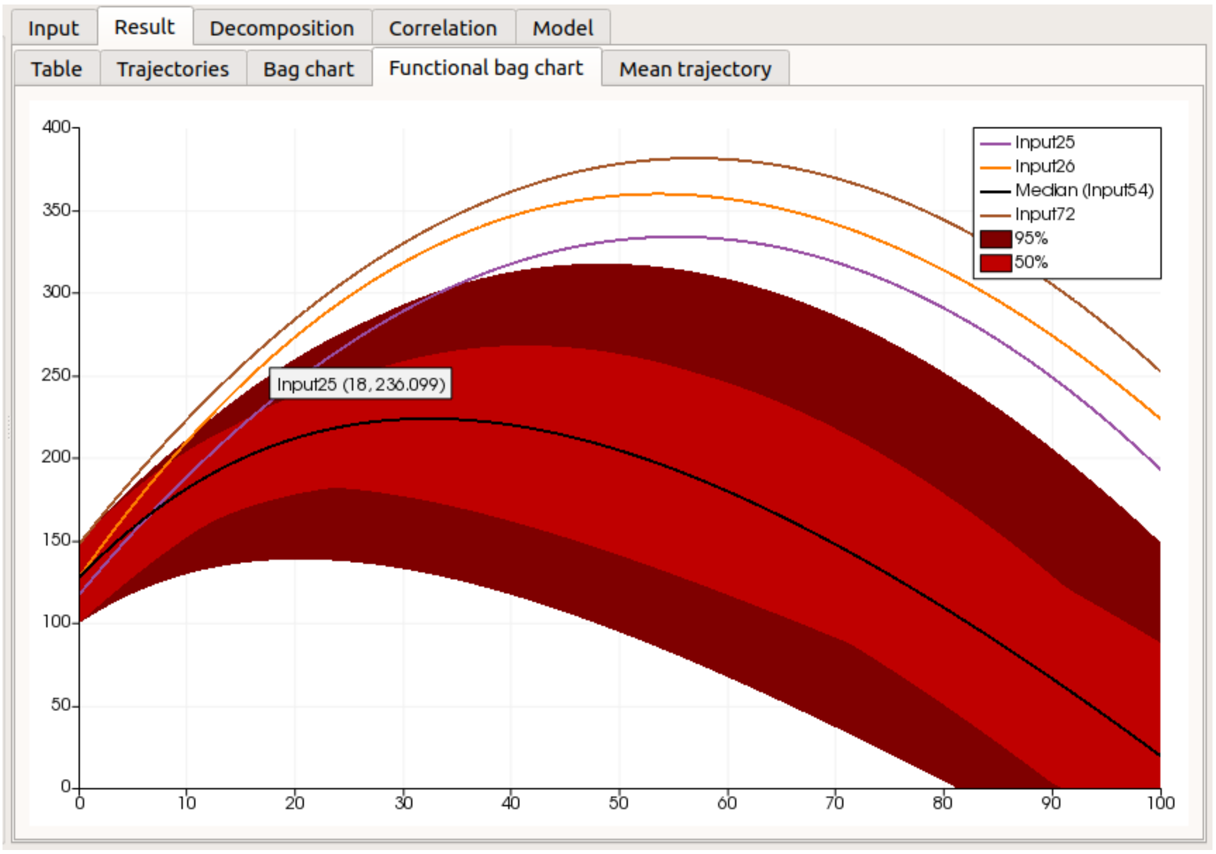


Figure 13: The functional bag chart of Paraview to plot a functional boxplot and detect outlier trajectories.