

# OpenTURNS and its graphical interface

Michaël Baudin <sup>1</sup>   Thibault Delage <sup>1</sup>   Anne Dutfoy <sup>1</sup>  
Anthony Geay <sup>1</sup>   Ovidiu Mircescu <sup>1</sup>   Aurélie Ladier <sup>2</sup>  
Julien Schueller <sup>2</sup>   Thierry Yalamas <sup>2</sup>

<sup>1</sup>EDF R&D. 6, quai Watier, 78401, Chatou Cedex - France, michael.baudin@edf.fr

<sup>2</sup>Phimeca Engineering. 18/20 boulevard de Reuilly, 75012 Paris - France,  
yalamas@phimeca.com

25 June 2019, UNCECOMP 2019, Crete, Greece



# Contents

Introduction

New sequential algorithms

PERSALYS, the graphical user interface

What's next ?

OpenTURNS: [www.openturns.org](http://www.openturns.org)

# OpenTURNS

An Open source initiative for the Treatment of Uncertainties, Risks'N Statistics

- ▶ Multivariate probabilistic modeling including dependence
- ▶ Numerical tools dedicated to the treatment of uncertainties
- ▶ Generic coupling to any type of physical model
- ▶ Open source, LGPL licensed, C++/Python library

# OpenTURNS: [www.openturns.org](http://www.openturns.org)



**AIRBUS**



- ▶ Linux, Windows
- ▶ First release : 2007
- ▶ 5 full time developers
- ▶ Users  $\approx$  1000, mainly in France (208 000 Total Conda downloads)
- ▶ Project size (2018) : 720 classes, more than 6000 services

# OpenTURNS: content

## Data analysis

**Visual analysis:** QQ-Plot, Cobweb

**Fitting tests:** Kolmogorov, Chi2

**Multivariate distribution:** kernel smoothing (KDE), maximum likelihood

**Process:** covariance models, Welch and Whittle estimators

**Bayesian calibration:** Metropolis-Hastings, conditional distribution

## Reliability, sensitivity

**Sampling methods:** Monte Carlo, LHS, low discrepancy sequences

**Variance reduction methods:** importance sampling, subset sampling

**Approximation methods:** FORM, SORM

**Indices:** Spearman, Sobol, ANCOVA

**Importance factors:** perturbation method, FORM, Monte Carlo

## Probabilistic modeling

**Dependence modelling:** elliptical, archimedean copulas.

**Univariate distribution:** Normal, Weibull

**Multivariate distribution:** Student, Dirichlet, Multinomial, User-defined

**Process:** Gaussian, ARMA, Random walk.

**Covariance models:** Matern, Exponential, User-defined

## Functional modeling

**Numerical functions:** symbolic, Python-defined, user-defined

**Function operators:** addition, product, composition, gradients

**Function transformation:** linear combination, aggregation, parametrization

**Polynomials:** orthogonal polynomial, algebra

## Meta modeling

**Functional basis methods:** orthogonal basis (polynomials, Fourier, Haar, Soize Ghanem)

**Gaussian process regression:**

General linear model (GLM), Kriging

**Spectral methods:** functional chaos (PCE), Karhunen-Loeve, low-rank tensors

## Numerical methods

**Integration:** Gauss-Kronrod

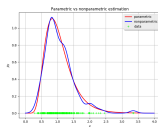
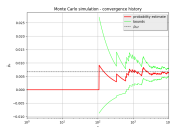
**Optimization:** NLOpt, Cobyla, TNC

**Root finding:** Brent, Bisection

**Linear algebra:** Matrix, HMat

**Interpolation:** piecewise linear, piecewise Hermite

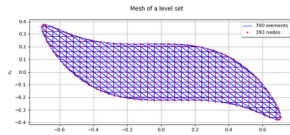
**Least squares:** SVD, QR, Cholesky



# OpenTURNS: documentation

## LevelSetMesher

(Source code, png, hires.png, pdf)



`class LevelSetMesher(*args)`

Creation of mesh of box type.

Available constructor:

`LevelSetMesher(discretization)`

Parameters: **discretization**: sequence of int, of dimension  $\leq 3$ .

Discretization of the levelset bounding box.

**solver**: `OptimizationAlgorithm`

Optimization solver used to project the vertices onto the level set. It must be able to solve nearest point problems. Default is `RobustKaczmarz`.

### Note

The meshing algorithm is based on the `IntervalMesher` class. First, the bounding box of the level set (provided by the user or automatically computed) is meshed. Then, all the simplices with all vertices outside of the level set are rejected, while the simplices with all vertices inside of the level set are kept. The remaining simplices are adapted the following way:

- The mean point of the vertices inside of the level set is computed
- Each vertex outside of the level set is projected onto the level set using a linear interpolation
- If the project flag is `True`, then the projection is refined using an optimization solver.

### Examples

Create a mesh:

```
>>> import openturns as ot
>>> mesher = ot.LevelSetMesher([5, 10])
>>> level = 1, 0
>>> function = ot.SymbolicFunction([x0', x1'], ['x0'^2+x1'^2'])
>>> levelSet = ot.LevelSetFunction(level)
>>> mesh = mesher.build(levelSet)
```

### Methods

|                                  |                                   |
|----------------------------------|-----------------------------------|
| <code>build(*args)</code>        | Build the mesh of level set type. |
| <code>getClassNames()</code>     | Accessor to the object's name.    |
| <code>getDiscretization()</code> | Accessor to the discretization.   |

- Content: programming interface, examples, theory.
- The doc is generated: *all* classes and methods are documented, partly automatically.
- Examples are automatically tested at *each* update of the code and outputs are checked.

## OpenTURNS: estimate the mean sequentially

Two sequential algorithms based on asymptotic statistics: the mean and Sobol' sensitivity indices.

Part 1 : Estimate the mean with an sequential algorithm.

- ▶ The "classical" way of estimating the mean : set the sample size  $n$ , then use the sample mean  $\bar{\mu} = (1/n) \sum_{j=1}^n y^{(j)}$  and estimate the accuracy (e.g. C.V.).
- ▶ Goal: use the smallest possible sample which achieves a given accuracy. Increase the sample size until a stopping criteria is met.
- ▶ The sample mean is asymptotically gaussian:

$$\bar{\mu} \xrightarrow{D} \mathcal{N} \left( E(Y), \frac{V(Y)}{n} \right).$$

- ▶ The absolute accuracy of the estimate  $\bar{\mu}$  can be evaluated based on the sample standard deviation of the estimator  $\hat{s}/\sqrt{n}$
- ▶ To get good performances on distributed supercomputers and multi-core workstations, the size of the sample increases by block.

# OpenTURNS: estimate the mean sequentially

```
[... Define the Y RandomVector ...]
algo = ot.ExpectationSimulationAlgorithm(Y)
algo.setMaximumOuterSampling(1000)
algo.setBlockSize(10) # Sample size is 0, 10, 20, 30, 40, ...
algo.setMaximumCoefficientOfVariation(0.001)
algo.run()
result = algo.getResult()
expectation = result.getExpectationEstimate()
print("Mean = %f" % expectation[0])
meanDistr = result.getExpectationDistribution()
View(meanDistr.drawPDF())
```

Output:

Mean = -5.972516



Asymptotic distribution of the sample mean.



# OpenTURNS: estimate Sobol' indices sequentially

Part 2 : Estimate Sobol' sensitivity indices with an incremental algorithm based on asymptotic statistics, extending the work of (Janon et al., 2014).

- ▶ Assume that the Sobol' estimator is:

$$\bar{S} = \Psi(\bar{U})$$

where  $\Psi$  is a multivariate function,  $U$  is a multivariate sample and  $\bar{U}$  is its sample mean.

- ▶ Each Sobol' estimator (e.g. Saltelli, Jansen, etc...) can be associated with a specific choice of function  $\Psi$  and vector  $U$ .
- ▶ Therefore, the multivariate delta method implies:

$$\sqrt{n}(\bar{U} - \mu) \xrightarrow{D} \mathcal{N}(0, \nabla\psi(\mu)^T \Gamma \nabla\psi(\mu))$$

where  $\mu$  is the expected value of the Sobol' indice,  $\nabla\psi(\mu)$  is the gradient of the function  $\Psi$  and  $\Gamma$  is the covariance matrix of  $\bar{U}$ .

- ▶ An implementation of the exact gradient  $\nabla\psi(\mu)$  was derived for all estimators in OpenTURNS (Dumas, 2018).

# OpenTURNS: estimate Sobol' indices sequentially

```
[... Define the X Distribution , define the g Function ...]
estimator = ot.SaltelliSensitivityAlgorithm()
estimator.setUseAsymptoticDistribution(True)
algo = ot.SobolSimulationAlgorithm(X, g, estimator)
algo.setMaximumOuterSampling(100) # number of iterations
algo.setBlockSize(50) # size of experiment at each iteration
algo.setIndexQuantileLevel(0.1) # the confidence interval level
algo.setIndexQuantileEpsilon(0.2) # length of confidence interval
algo.run()
```



Asymptotic distribution of the first order Sobol' indices for the first variable.

# PERSALYS, the graphical user interface of OpenTURNS

- ▶ Main goal : provide a graphical interface of OpenTURNS in the SALOME integration platform
- ▶ Features
  - ▶ Uncertainty quantification : definition of the probabilistic model (including dependence), distribution fitting (including copulas), central tendency, sensitivity analysis, probability estimate, meta-modeling (polynomial chaos, kriging), screening with Morris, optimization, design of experiments
  - ▶ Generic (not dedicated to a specific application)
  - ▶ GUI language : English, French
- ▶ Partners : EDF, Phiméca
- ▶ Licence : LGPL
- ▶ Schedule :
  - ▶ Since summer 2016, one EDF release per year
  - ▶ On the internet (free) : SALOME\_EDF since 2018 on <https://www.salome-platform.org>

# PERSALYS: define the dependence

- ▶ Dependence is defined using copulas
- ▶ Define arbitrary groups of dependent variables
- ▶ Available copulas (same as in OT): gaussian, Ali-Mikhail-Haq, Clayton, Farlie-Gumbel-Morgenstern, Frank, Gumbel
- ▶ Dependence inference from a sample : Bayesian Information Criteria (BIC) or Kendall plot



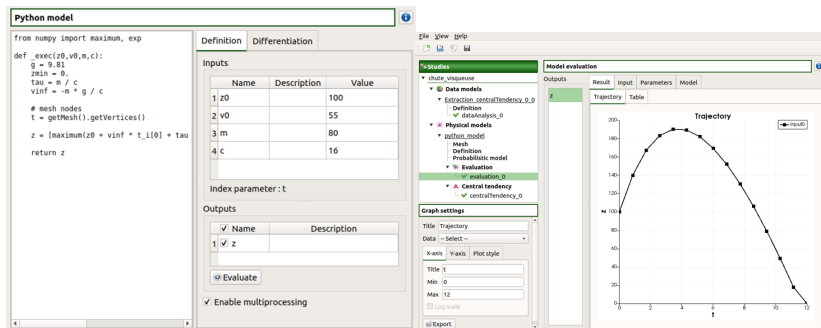
# PERSALYS: 1D fields

- ▶ Mesh definition and visualization
- ▶ Import from text or csv file



# PERSALYS: 1D fields

- ▶ Functional model definition and probabilistic model
- ▶ Python or symbolic



# PERSALYS: 1D fields

- ▶ Probabilistic model
- ▶ Uncertainty propagation with simple Monte-Carlo sampling



# PERSALYS: 1D fields

- ▶ BagChart and Functional Bagchart (from Paraview) based on High Density Regions (Hyndman, 1996).
- ▶ To do this, Paraview uses a principal component analysis decomposition.
- ▶ Linked and interactive selections in the views.





# What's next ?

## PERSALYS Roadmap :

- ▶ Calibration
- ▶ 2D Fields, 3D Fields
- ▶ In-Situ fields based on the MELISSA library (with INRIA):  
when we cannot store the whole sample in memory or on the hard drive, update the statistics (e.g. the mean, Sobol' indices) sequentially, with distributed computing.



# The end

Thanks !

Questions ?

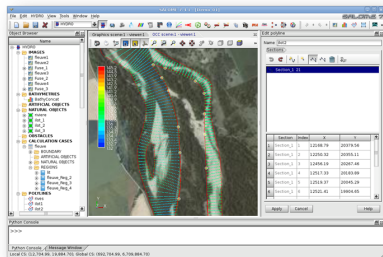
# PERSALYS: estimate the parameters of the copulas

- Inference of the dependence of the multivariate sample
- Guided choice according to the BIC and Kendall plot



# SALOME

- ▶ Integration platform for pre and post processing, and 2D/3D numerical simulation
- ▶ Features : geometry, mesh, distributed computing
- ▶ Visualization, data assimilation, uncertainty treatment
- ▶ Partners : EDF, CEA, Open Cascade
- ▶ Licence : LGPL
- ▶ Linux, Windows
- ▶ [www.salome-platform.org](http://www.salome-platform.org)



# OpenTURNS: estimate Sobol' indices sequentially

Part 2 : Estimate Sobol' sensitivity indices with an incremental algorithm.

- ▶ Let us denote by  $\Phi_k^F$  (resp.  $\Phi_k^T$ ) the cumulated distribution function of the gaussian distribution of the first (resp. total) order sensitivity indice of the k-th input variable.
- ▶ We set  $\alpha \in [0, 1]$  a quantile level and  $\epsilon \in (0, 1]$  a quantile precision.
- ▶ The algorithms stops when, on all components, first and total order indices have been estimated with enough precision.

The precision is said to be sufficient if

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

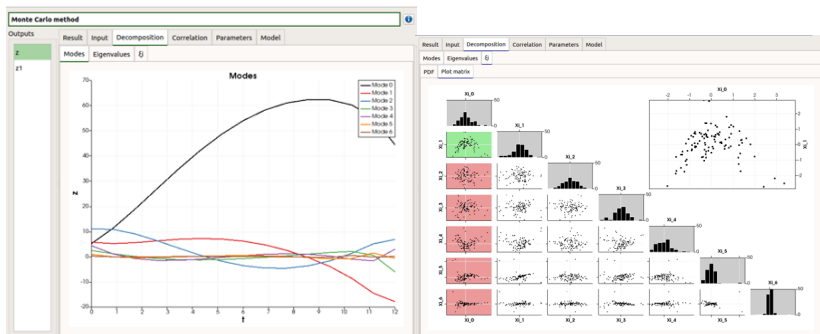
and

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

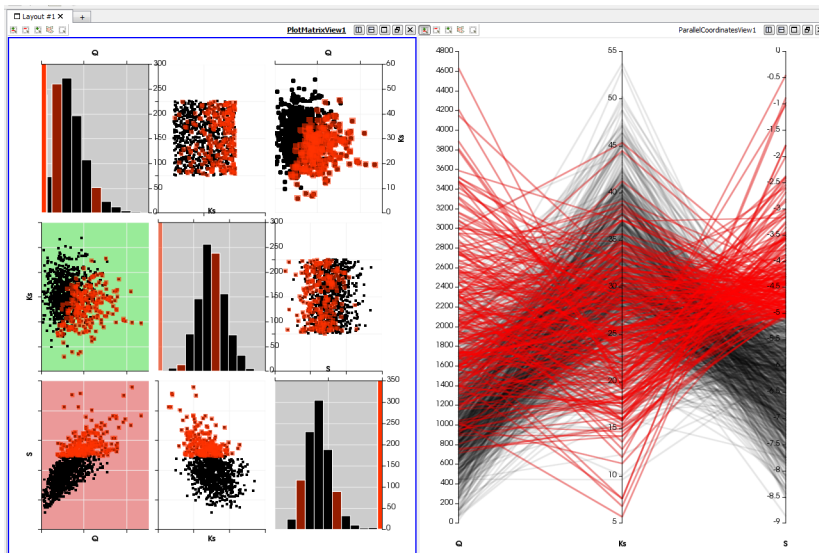
for  $k = 1, \dots, n_X$ .

# PERSALYS: 1D fields

- ▶ Karhunen Loeve decomposition
- ▶ Show modes, eigenvalues and projection coefficients



# Interactive uncertainty visualization with Paraview



# Methodology





# Software architecture

Two entry points:

- ▶ interactive,
- ▶ Python.

Advantages of the Python programming of the GUI:

- ▶ unit tests,
- ▶ going beyond the GUI



# Symbolic physical model

The screenshot shows the OTGui application window. On the left is a tree view under 'Crue Analytique' containing 'physicalModel\_0', which has sub-items for 'Deterministic study', 'Probabilistic study', and 'Designs of experiment'. The main area is divided into 'Inputs' and 'Outputs' sections.

**Inputs**

|   | Name | Description                     | Value |
|---|------|---------------------------------|-------|
| 1 | Q    | Débit (m <sup>3</sup> /s)       | 0     |
| 2 | Ks   | Strickler (m <sup>1/3</sup> /s) | 30    |
| 3 | Zv   | Côte aval (m)                   | 50    |
| 4 | Zm   | Côte amont (m)                  | 55    |
| 5 | Hd   | Hauteur de la digue (m)         | 8     |
| 6 | Zb   | Côte de la berge (m)            | 55,5  |
| 7 | L    | Longueur de la rivière (m)      | 5000  |
| 8 | B    | Largeur de la rivière (m)       | 300   |

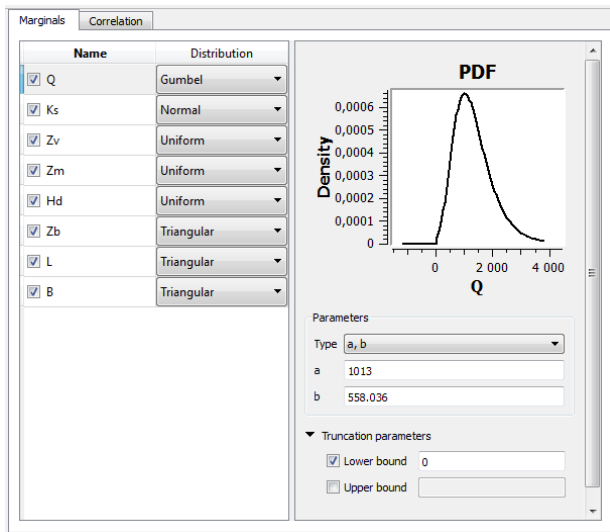
Buttons: + Add, - Remove

**Outputs**

|   | Name | Description  | Formula  | Value |
|---|------|--------------|--|-------|
| 1 | H    | Surverse (m) | $(Q / (Ks * B * \sqrt{(Zm - Zv) / L}))^{(3.0 / 5.0)} + Zv - Zb - Hd$ | -13,5 |

Buttons: + Add, - Remove, Evaluate

# Probabilistic model

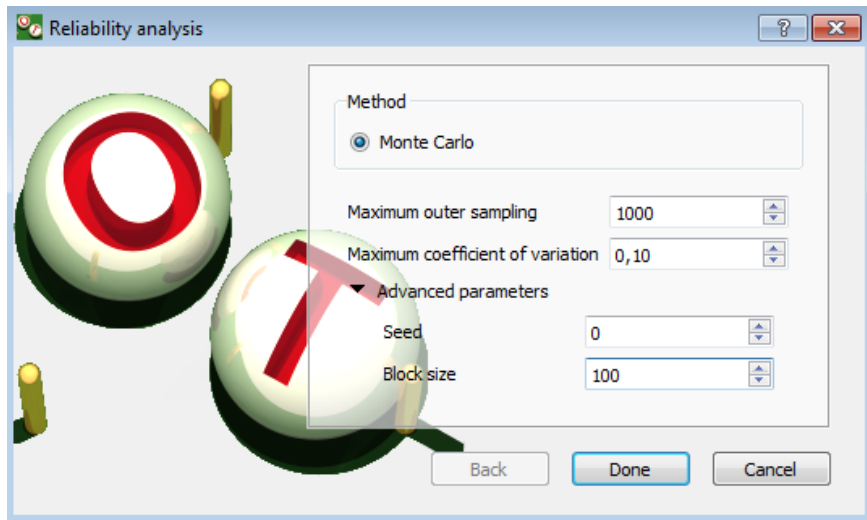


## Limit state study : definition of the threshold

Definition of the failure event :

| Output | Operator | Threshold |
|--------|----------|-----------|
| H ▼    | < ▼      | -10       |

# Limit state study : algorithm parameters



# Limit state study : summary

Summary

Histogram

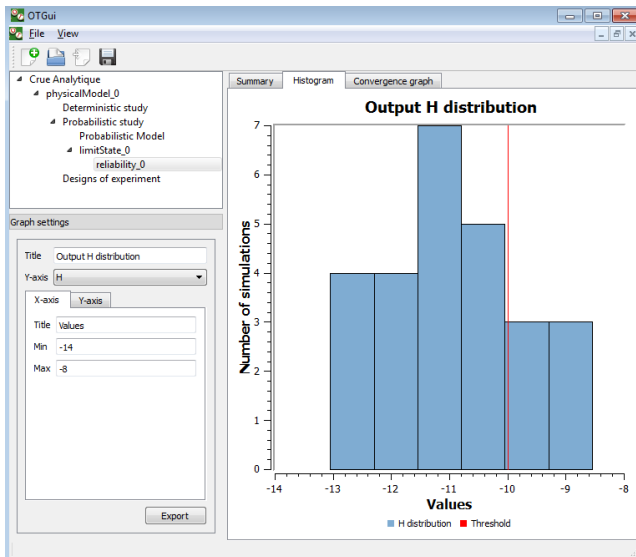
Convergence graph

Output H

Number of simulations: 26

| Estimate                 | Value     | Confidence interval at 95% |             |
|--------------------------|-----------|----------------------------|-------------|
|                          |           | Lower bound                | Upper bound |
| Failure probability      | 0.807692  | 0.656203                   | 0.959182    |
| Coefficient of variation | 0.0956949 |                            |             |

# Limit state study : histogram



# Central tendency : algorithm parameters





# Central tendency : summary results

Moments estimate

| Estimate           | Value    | Confidence interval at 95% |             |
|--------------------|----------|----------------------------|-------------|
|                    |          | Lower bound                | Upper bound |
| Mean               | -11.0178 | -11.0417                   | -10.9938    |
| Standard deviation | 1.22309  | 1.20637                    | 1.24028     |
| Skewness           | 0.20005  |                            |             |
| Kurtosis           | 3.01907  |                            |             |
| First quartile     | -11.8721 |                            |             |
| Third quartile     | -10.2129 |                            |             |

**Probability**   
**Quantile**

# Central tendency : summary results

| Result table                 |          |          |          |
|------------------------------|----------|----------|----------|
| Summary                      |          |          |          |
| PDF/CDF                      |          |          |          |
| Box plots                    |          |          |          |
| Scatter plots                |          |          |          |
| Output <span>H ▾</span>      |          |          |          |
| Number of simulations: 10000 |          |          |          |
| Minimum and Maximum          |          |          |          |
|                              | Variable | Minimum  | Maximum  |
| Inputs at extremum           | Output   | -15.0155 | -5.88758 |
|                              | Q        | 7.97827  | 6187.43  |
|                              | Ks       | 27.132   | 25.5926  |
|                              | Zv       | 49.1681  | 50.9071  |
|                              | Zm       | 54.5469  | 55.3994  |
|                              | Hd       | 8.76082  | 8.49391  |
|                              | Zb       | 55.5436  | 55.4935  |
|                              | L        | 4999.26  | 4997.37  |
|                              | B        | 303.187  | 300.871  |

# Central tendency : scatter plots



# Sensitivity analysis : Sobol' indices



# OpenTURNS: estimate the mean

See the Jupyter Notebook.

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

```
ot.RandomGenerator.SetSeed(0)
```

*# 1. The function G*

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

*# Creation of the problem function*

```
g = ot.PythonFunction(4, 1, functionCrue)
g = ot.MemoizeFunction(g)
```

# OpenTURNS: estimate the mean

## # 2. Random vector definition

```
myParamQ = ot.GumbelAB(1013., 558.)
Q = ot.ParametrizedDistribution(myParamQ)
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)
```

## # 3. View the PDF

```
Q.setDescription(["Q□ (m3/s)"])
View(Q.drawPDF()).show()
```



# OpenTURNS: estimate the mean

```
# 4. Create the joint distribution function ,
# the output and the event.
X = ot.ComposedDistribution([Q, Ks, Zv, Zm])
Y = ot.RandomVector(g, ot.RandomVector(X))

# 5. Estimate expectation with simple Monte-Carlo
sampleSize = 10000
sampleX = X.getSample(sampleSize)
sampleY = g(sampleX)
sampleMean = sampleY.computeMean()
print ("Mean=%f" % (sampleMean[0]))
```

Output:

Mean by MC = -5.937845

# GUI : the demo

Demo time.



## GUI : outline

- ▶ From scratch : 3 inputs, 2 outputs, sum, central dispersion study with default parameters
- ▶ Open axialStressedBeam-python.xml : central dispersion with sample size 1000, Threshold  $P(G < 0)$  with  $CV=0.05$
- ▶ Import crue-4vars-analytique.py : S.A. with sample size 1000, sort by size

## UQ, the easy way

Main goal : make UQ easy to use

- ▶ classical user-friendly algorithms with a state-of-the-art implementation,
- ▶ default parameters of the algorithms whenever possible,
- ▶ an easy access to the HPC resources,
- ▶ an automated connection to the computer code.

Produce standard results :

- ▶ numerical results e.g. tables,
- ▶ classical graphics.

## Overview (1/2)

Inputs from the user :

- ▶ Physical model : symbolic, Python code or SALOME component
- ▶ Probabilistic model : joint probability distribution function of the input.

Then :

- ▶ Central dispersion: estimates the central dispersion of the output  $Y$  (e.g. mean).
- ▶ Threshold probability: estimates the probability that the output exceeds a given threshold  $S$ .
- ▶ Sensitivity analysis: estimates the importance of the inputs to the variability of the output.

## Overview (2/2)

Probabilistic modeling :

- ▶ Distribution fitting from a sample
- ▶ Dependence modeling (Gaussian copula)

Meta-modeling :

- ▶ Polynomial chaos (full or sparse)
- ▶ Kriging