OpenTURNS and its graphical interface

Michaël Baudin [1] Anne Dutfoy [1] Anthony Geay [1] Anne-Laure Popelin [1] Aurélie Ladier [2] Julien Schueller [2] 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

15 June 2017, UNCECOMP 2019, Rhodes, Greece

Contents

OpenTURNS: 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

- Linux, Windows
- First release : 2007
- 4 full time developers
- Users ≈ 1000, mainly in France (10900 Conda downloads in 2016-2017)
- 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

Probabilistic modeling

Dependence modelling: elliptical, archimedian copulas.
Univariate distribution: Normal, Weibull
Multivariate distribution: Student, Dirichlet, Multinomial, User-defined
Process: Gaussian, ARMA, Random walk.
Covariance models: Matern, Exponential, User-defined

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

Reliability, sensitivity

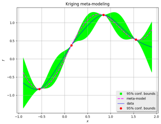
Sampling methods: Monte Carlo, LHS, low discrepancy sequences
Variance reduction methods: importance sampling, subset sampling
Approximation methods: FORM, SORM
Indices: Spearman, Sobol, ANCOVA
Importance factors: perturbation method, FORM, Monte Carlo

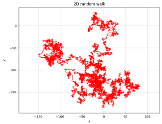
Functional modeling

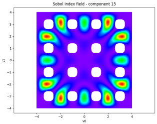
Numerical functions: symbolic, Python-defined, user-defined
Function operators: addition, product, composition, gradients
Function transformation: linear combination, aggregation, parametrization
Polynomials: orthogonal polynomial, algebra

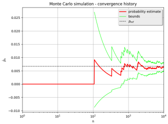
Numerical methods

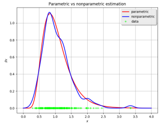
Integration: Gauss-Kronrod
Optimization: NLopt, Cobyla, TNC
Root finding: Brent, Bisection
Linear algebra: Matrix, HMat
Interpolation: piecewise linear, piecewise Hermite
Least squares: SVD, QR, Cholesky









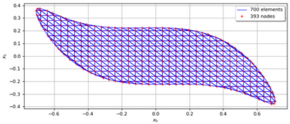


OpenTURNS: documentation

LevelSetMesher

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

Mesh of a level set



class **LevelSetMesher**(**args*)

Creation of mesh of box type.

Available constructor:

LevelSetMesher(*discretization*)

Parameters: **discretization** : sequence of int, of dimension ≤ 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 `AbdoRackwitz`.

Notes

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 levelset 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.LevelSet(function, level)
>>> mesh = mesher.build(levelSet)
```

Methods

`build`(*args)	Build the mesh of levelset type.
`getClassName`()	Accessor to the object's name.
`getDiscretization`()	Accessor to the discretization.

Flood model

We consider the test case of the overflow of a river:

$$S = \left(\frac{Q}{Ks \times 300 \times \sqrt{(Zm - Zv)/5000}} \right)^{(3/5)+Zv-55.5-3}$$

It is considered that failure occurs when the difference between the dike height and the water level is positive:

$$S > 0$$

Four independent random variables are considered:

- Q (Flow rate) [m^3 s^-1]
- Ks (Strickler) [m^1/3 s^-1]
- Zv (downstream height) # [m]
- Zm (upstream height) # [m]

Stochastic model:

- Q ~ Gumbel(alpha=0.00179211, beta=1013), Q > 0
- Ks ~ Normal(mu=30.0, sigma=7.5), Ks > 0
- Zv ~ Uniform(a=49, b=51)
- Zm ~ Uniform(a=54, b=56)

In [52]:
```
from __future__ import print_function
import openturns as ot
```

In [53]:
```
# Create the marginal distributions of the parameters
dist_Q = ot.TruncatedDistribution(ot.Gumbel(1./558., 1013.), 0, ot.TruncatedDistributi
dist_Ks = ot.TruncatedDistribution(ot.Normal(30.0, 7.5), 0, ot.TruncatedDistribution.L
dist_Zv = ot.Uniform(49.0, 51.0)
dist_Zm = ot.Uniform(54.0, 56.0)
marginals = [dist_Q, dist_Ks, dist_Zv, dist_Zm]
```

In [54]:
```
# Create the Copula
RS = ot.CorrelationMatrix(4)
#RS[2, 3] = -0.2
# Evaluate the correlation matrix of the Normal copula from RS
R = ot.NormalCopula.GetCorrelationFromSpearmanCorrelation(RS)
```

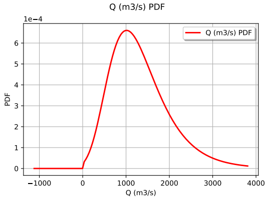
OpenTURNS: estimate the mean

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

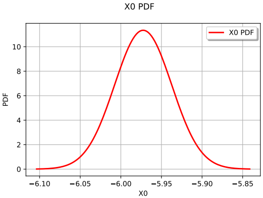
```
# 4. Create the joint distribution function,
#    the output and the event.
X = ot.ComposedDistribution([Q, Ks, Zv, Zm])
Y = ot.RandomVector(g, ot.RandomVector(X))

# 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
```

OpenTURNS: estimate the mean

```
# 6. Estimate expectation with algorithm
algo = ot.ExpectationSimulationAlgorithm(Y)
algo.setMaximumOuterSampling(1000)
algo.setBlockSize(1)
algo.setCoefficientOfVariationCriterionType('N
algo.run()
result = algo.getResult()
expectation = result.getExpectationEstimate()
print("Mean␣by␣ESA␣=␣%f␣" % expectation[0])
expectationDistribution = result.getExpectatic
View(expectationDistribution.drawPDF())
```

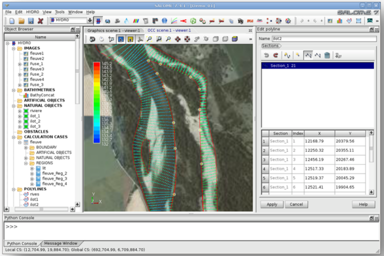


Output:

Mean by ESA = −5.972516

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`



The graphical user interface of OpenTURNS

- Main goal : provide a graphical interface of OpenTURNS in SALOME
- Features
  - Uncertainty quantification (distribution fitting), central tendency, sensitivity analysis, probability estimate, meta-modeling
  - 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 : 2018

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)

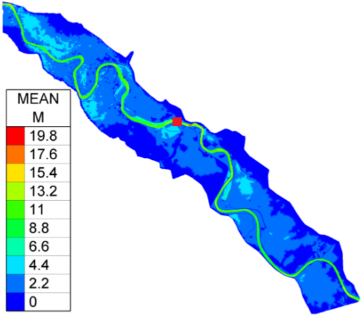
Meta-modeling :

- Polynomial chaos (full or sparse)
- Kriging

Fields

Field example :

- Input : 4 independent random variables
- Output : height of the river Garonne on a 100 km segment
- Computer code : TELEMAC2D
- Quantity of interest : pointwise average over 70 000 random simulations



Roadmap :

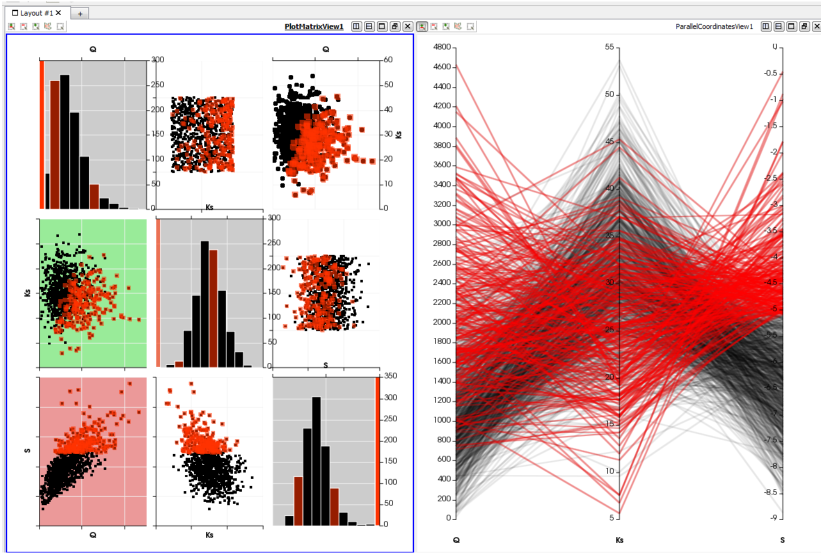
- Now : massive Python/OpenTURNS scripting
- 2017-2018 : in the gui

The end

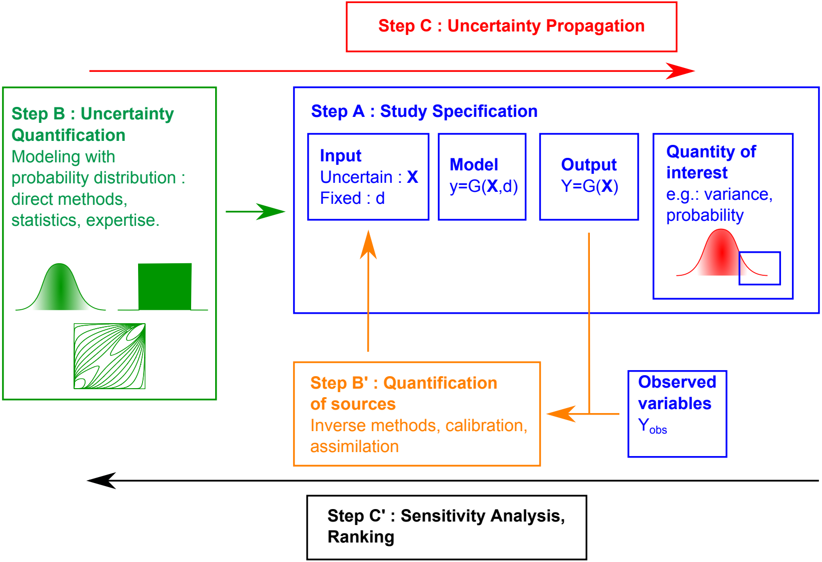
Thanks !

Questions ?

Interactive uncertainty visualization with Paraview



Methodology



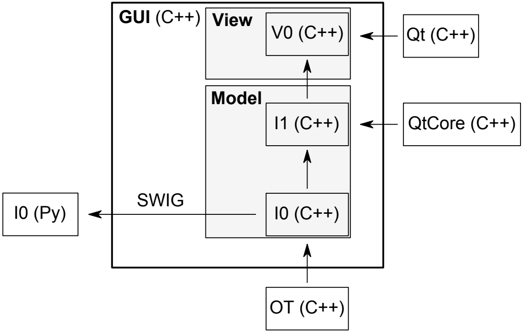
Software architecture

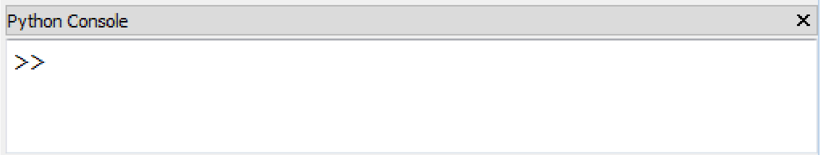
Two entry points:

- interactive,
- Python.

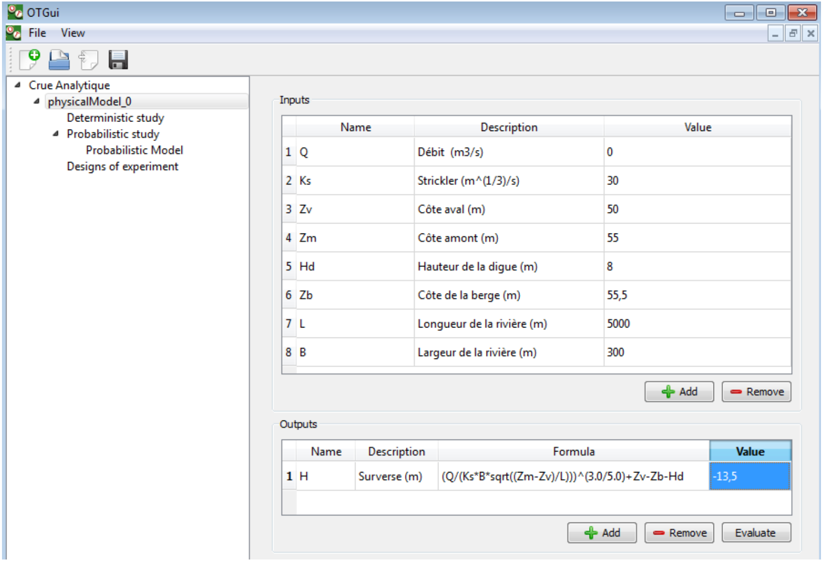
Advantages of the Python programming of the GUI:

- unit tests,
- going beyond the GUI

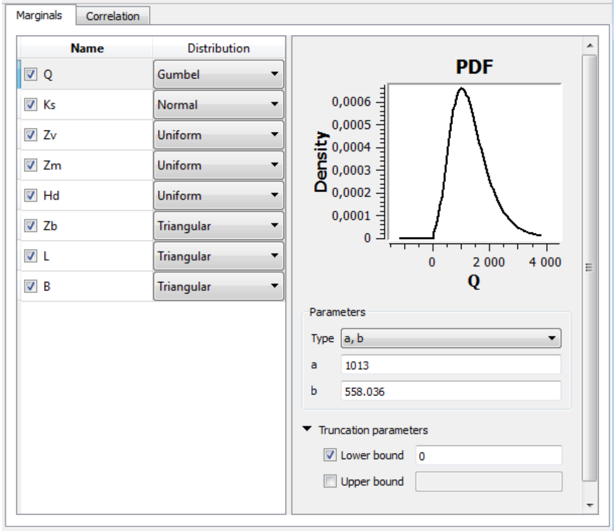




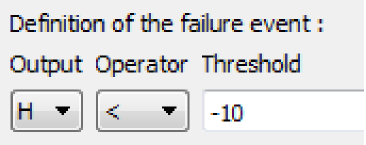
Symbolic physical model



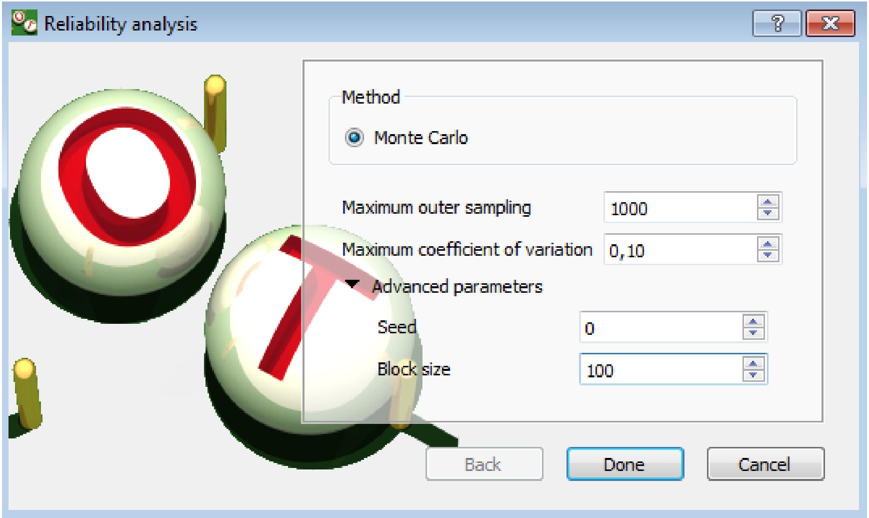
Probabilistic model



Limit state study : definition of the threshold



Limit state study : algorithm parameters



Limit state study : summary

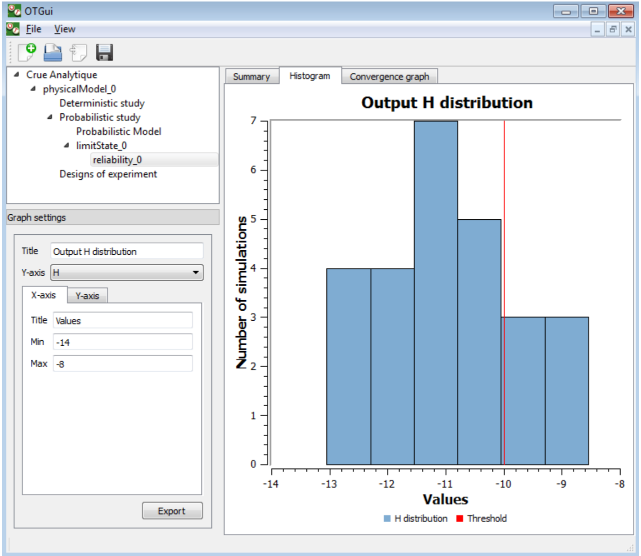
Summary | Histogram | Convergence graph

Output H

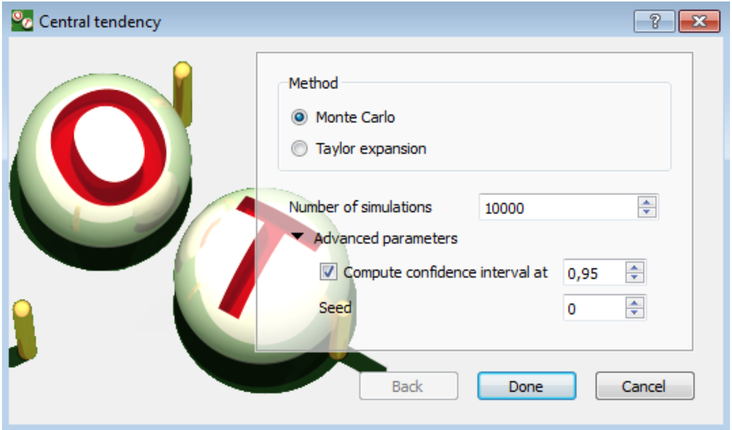
Number of simulations: 26

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 0,50 **Quantile** -11,04288948

Central tendency : summary results

Result table | Summary | PDF/CDF | Box plots | Sca

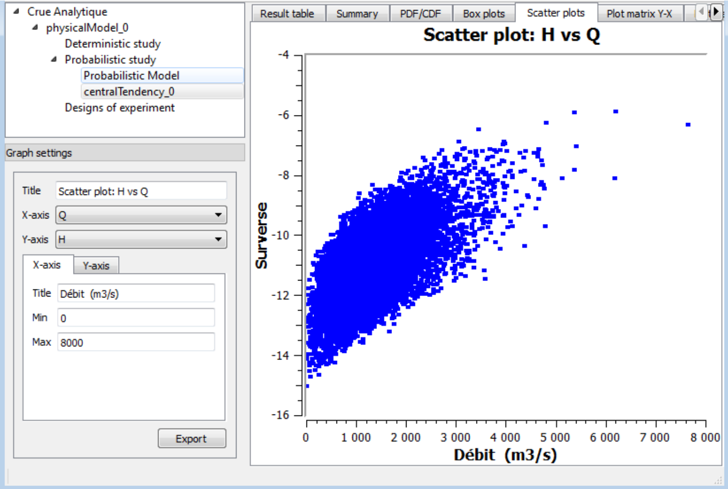
Output H

Number of simulations: 10000

Minimum and Maximum

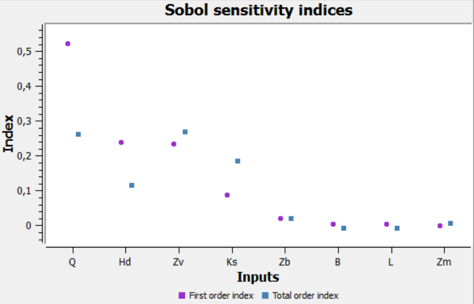
	Variable	Minimum	Maximum
Output	H	-15.0155	-5.88758
Inputs at extremum	Q	7.97827	6187.43
	Ks	27.132	25.5926
	Zv	49.1681	50.9071
	Zm	54.5469	55.3994
	Hd	8.76082	8.49391
	Zb	55.5436	55.4935
	L	4999.26	4997.37
	B	303.187	300.871

Central tendency : scatter plots



Sensitivity analysis : Sobol' indices

Result



Input	First order index	Total order index
Zv	0.2366	0.2707
Q	0.5246	⚠ 0.2646
Ks	0.08843	0.1881
Hd	0.2411	⚠ 0.1179
Zb	0.02115	0.02203
Zm	0.0002773	0.008043

Interactions -2.784 ⚠