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] 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

25 June 2019, UNCECOMP 2019, Crete, Greece

2019-06-23

OpenTURNS

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] 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

25 June 2019, UNCECOMP 2019, Crete, Greece

I would like to thank the organizers to inviting us to present our tools.

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

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 is a software for uncertainty quantification, uncertainty propagation, sensitivity analysis and metamodeling.
The software is available with the open source LGPL licence on Linux and Windows.

In order to use OpenTURNS, you can use directly the C++ library, or program your Python scripts.

OpenTURNS: www.openturns.org

- 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: www.openturns.org

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

It is developped by five parterns : EDF, Phiméca, Airbus, IMACS and ONERA.
OpenTURNS was first released in two thousands and seven.
It is developped by 5 full time developers.
There are approximately one thousand users worlwide, mainly in France.
This is a significant software project, with seven hundred classes, more than 300 000 (three hundred thousands) lines of codes.

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

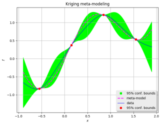
Sampling methods: Monte Carlo, LHS, low discrepancy sequences
Variance reduction methods: importance sampling, subset sampling
Approximation methods: FORM, SORM
Indices: Spearman, Sobol, ANCOVA
Importance factors: perturbation method, FORM, Monte Carlo

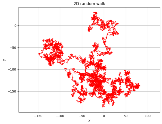
Functional modeling

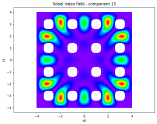
Numerical functions: symbolic, Python-defined, user-defined
Function operators: addition, product, composition, gradients
Function transformation: linear combination, aggregation, parametrization
Polynomials: orthogonal polynomial, algebra

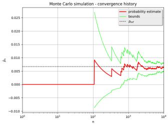
Numerical methods

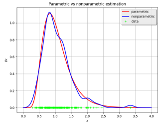
Integration: Gauss-Kronrod
Optimization: NLopt, Cobyla, TNC
Root finding: Brent, Bisection
Linear algrebra: Matrix, HMat
Interpolation: piecewise linear, piecewise Hermite
Least squares: SVD, QR, Cholesky

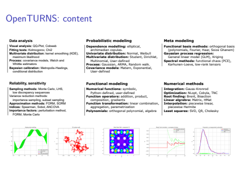












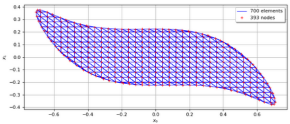
This is a global picture of the main features: data analysis, probabilistic modeling, meta-modeling, reliability, sensitivity analysis, functional modeling and numerical methods, and, of course, graphics.

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 level set 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 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.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]

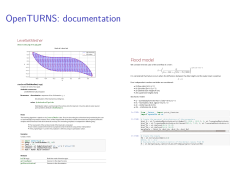
Stochastic model:

- 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)
```



The documentation is available online with a technical documentation of the programming interface, that is to say, the classes (for example the LevelSetMesher class) with a description of the arguments and small examples (the examples are automatically tested).

The documentation also provide larger examples and a theoretical documentation.

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 $\mu = (1/n) \sum_{j=1}^{n} y^{(j)}$.
- The sample mean is asymptotically gaussian:

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

- The absolute precision of the estimate μ can be evaluated based on the sample standard deviation of the estimator $\frac{s}{\sqrt{n}}$
- To set the relative precision, we can consider the coefficient of variation of the estimator (if $E(Y) \neq 0$).
- To get good performances on distributed supercomputers and multi-core workstations, the size of the sample increases by block.

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 $\mu = (1/n)\sum_{j=1}^{n} y^{(j)}$.
- The sample mean is asymptotically gaussian:

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

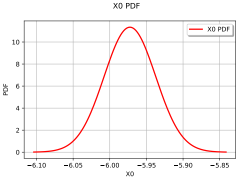
- The absolute precision of the estimate μ can be evaluated based on the sample standard deviation of the estimator $\frac{s}{\sqrt{n}}$
- To set the relative precision, we can consider the coefficient of variation of the estimator (if $E(Y) \neq 0$).
- To get good performances on distributed supercomputers and multi-core workstations, the size of the sample increases by block.

I would now like to introduce two new sequential algorithms which are new in OT 1.12.
These algorithms are based on asymptotic statistics.
Part 1 is on estimating the mean and part 2 is on sensitivity indices.
Suppose that you want to estimate the mean of a random variable Y, where Y is the output of a computer code.
Assume that there is only one single output (it is easy to generalize this to multiple outputs).

OpenTURNS: estimate the mean sequentially

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

Mean = −5.972516

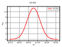


Asymptotic distribution of the sample mean.

OpenTURNS: estimate the mean sequentially

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

Mean = −5.972516



Asymptotic distribution of the sample mean.

We use the ExpectationSimulationAlgorithm class.
We set the setMaximumOuterSampling to 1000 (one thousands). This is the number of outer iterations.
We set the setBlockSize to 10 (ten). This is the block size (i.e. the number of inner iterations).
To define the stopping criteria, we set the setMaximumCoefficientOfVariation to 0.001 (point zero zero one).
And we run.
The getResult method returns an object that contains the result of the algorithm. The getExpectationEstimate returns the estimated mean.
Most importantly, the getExpectationDistribution method returns the gaussian distribution, that we can plot. We can, if required, derive a 95% confidence interval.

OpenTURNS: estimate Sobol' indices sequentially

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

- 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 (e.g. Saltelli, Jansen, etc...) can be associated with a specific choice of function Ψ and vector U.
- Therefore, the multivariate delta method implies:

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

  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.

OpenTURNS: estimate Sobol' indices sequentially

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

- 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 (e.g. Saltelli, Jansen, etc...) can be associated with a specific choice of function Ψ and vector U.
- Therefore, the multivariate delta method implies:

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

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.

We did the same to estimate the sensitivity indices.
The trick is to define the Sobol' estimator as follows ... where Ψ is a multivariate function, U is a multivariate sample and $\overline{U}$ is its sample mean.

OpenTURNS: estimate Sobol' indices sequentially

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

- Let us denote by Φ_k^F (resp. Φ_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 haved 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 $k = 1, ..., n_X$.

OpenTURNS: estimate Sobol' indices sequentially

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

- Let us denote by Φ_k^F (resp. Φ_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 haved 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

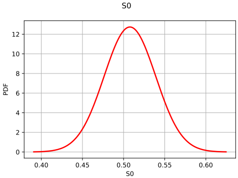
$$\Phi_k^T(1-\alpha) - \Phi_k^T(\alpha) \leq \epsilon$$

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

We now define the stopping criteria.

OpenTURNS: estimate Sobol' indices sequentially

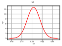
```
[... Define the X Distribution, define the g Function...]
estimator = ot.SaltelliSensitivityAlgorithm()
estimator.setUseAsymptoticDistribution(True)
algo = ot.SobolSimulationAlgorithm(X, g, estimator)
algo.setMaximumOuterSampling(100) # number of iterations
algo.setBlockSize(50)  # size of Sobol experiment at each iteration
algo.setBatchSize(16) # number of points evaluated simultaneously
# alpha, the confidence interval level
algo.setIndexQuantileLevel(0.1)
# epsilon, a quantile precision
algo.setIndexQuantileEpsilon(0.2)
algo.run()
```



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

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 Sobol experiment at each iteration
algo.setBatchSize(16) # number of points evaluated simultaneously
# alpha, the confidence interval level
algo.setIndexQuantileLevel(0.1)
# epsilon, a quantile precision
algo.setIndexQuantileEpsilon(0.2)
algo.run()
```



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

We use the SaltelliSensitivityAlgorithm class to define the sensitivity estimator.

The sequential algorithm is provided by the SobolSimulationAlgorithm class.

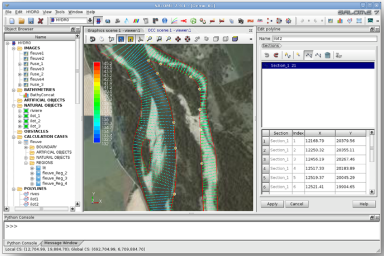
We set the setMaximumOuterSampling, which is the number of outer iterations.

We set the block size with setBlockSize (i.e. the size of the inner iterations).

Then the setIndexQuantileLevel sets the confidence interval level and the setIndexQuantileEpsilon sets the confidence interval length.

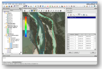
SALOME

- 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`



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`



Before presenting the graphical user interface, I would like to say some words about SALOME.

PERSALYS, the graphical user interface of OpenTURNS

- Main goal : provide a graphical interface of OpenTURNS in SALOME
- 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 : SALOME_EDF since 2018 on `https://www.salome-platform.org` (as free as a free beer)

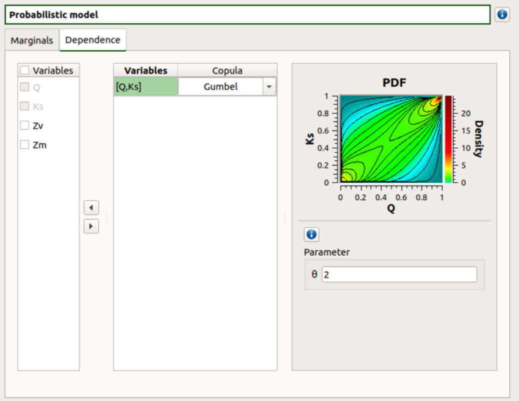
PERSALYS, the graphical user interface of OpenTURNS

- Main goal : provide a graphical interface of OpenTURNS in SALOME
- 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 : SALOME_EDF since 2018 on `https://www.salome-platform.org` (as free as a free beer)

We created a new tool within SALOME, called PERSALYS.

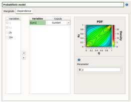
PERSALYS: define the dependence

- 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



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

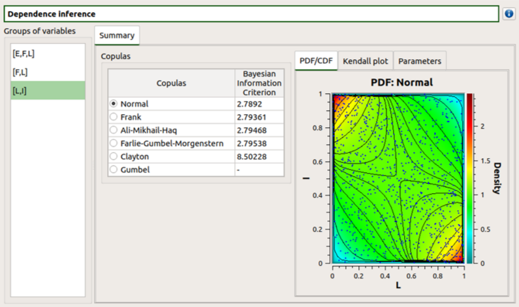


To define the dependence in PERSALYS, we use the copula tools available in OpenTURNS.

The Dependence tab shows the list of input variables: you can create a group of dependent variables using the checkbutton and add it to the group.

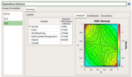
PERSALYS: estimate the parameters of the copulas

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



PERSALYS: estimate the parameters of the copulas

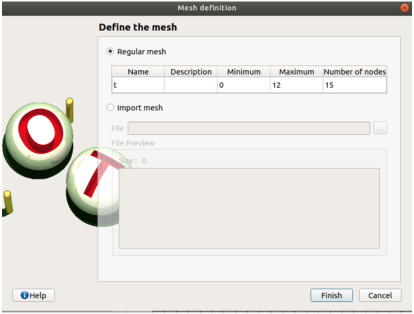
- Inference of the dependence of the multivariate sample
- Guided choice according to the BIC and Kendall plot

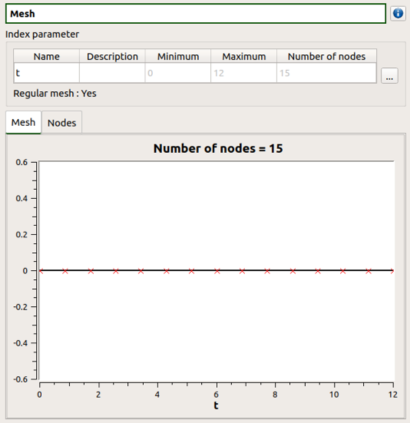


To estimate the parameters of a copula, you can perform the inference of a multivariate sample.

PERSALYS: 1D fields

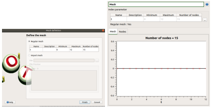
- Mesh definition and visualization
- Import from text or csv file





PERSALYS: 1D fields

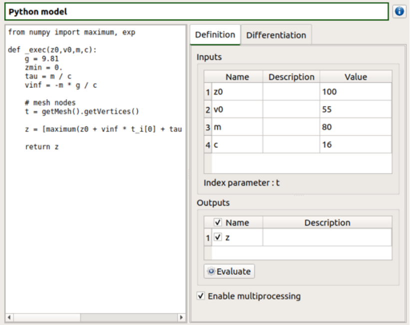
- Mesh definition and visualization
- Import from text or csv file

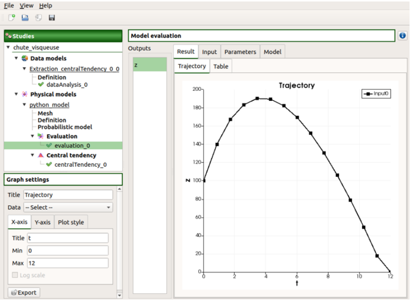


The most important new feature is the management of 1D stochastic processes.

PERSALYS: 1D fields

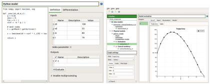
- Functional model definition and probabilistic model
- Python or symbolic





PERSALYS: 1D fields

- Functional model definition and probabilistic model
- Python or symbolic

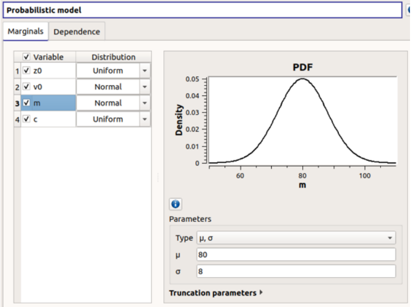


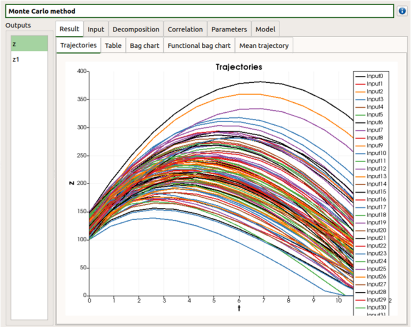
The Python function has 4 (four) inputs named z0, v0, m and c. Moreover, the Python function depends on the index parameter which name is "t".

On the right figure, this is what appears when we click on the "Evaluate" button: one evaluation of the function is then a trajectory which depends on the time.

PERSALYS: 1D fields

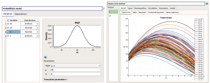
- Probabilistic model
- Uncertainty propagation with simple Monte-Carlo sampling





PERSALYS: 1D fields

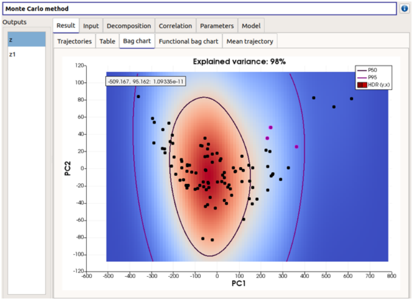
- Probabilistic model
- Uncertainty propagation with simple Monte-Carlo sampling

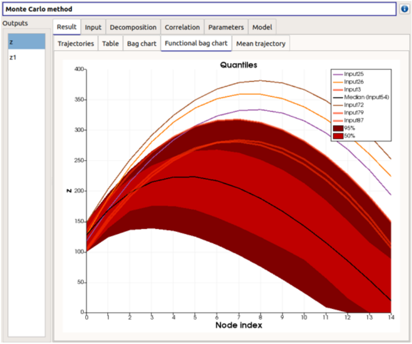


Then we can perform the central dispersion analysis of the model.
On the left, we define the distribution of the input random vector.
On the right, this is the result of the simulation: a sample made of 100 (one hundred) trajectories.

PERSALYS: 1D fields

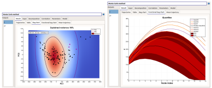
- BagChart and Functional Bagchart (from Paraview) based on High Density Regions.
- To do this, Paraview uses OpenTURNS to perform the Karhunen-Loève decomposition.
- Linked selections in the views





PERSALYS: 1D fields

- BagChart and Functional Bagchart (from Paraview) based on High Density Regions.
- To do this, Paraview uses OpenTURNS to perform the Karhunen-Loève decomposition.
- Linked selections in the views

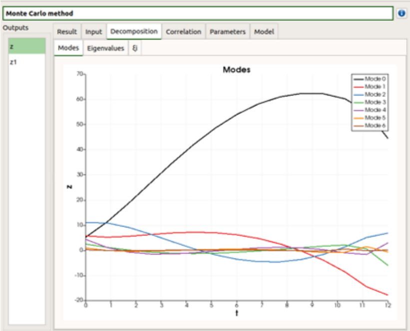


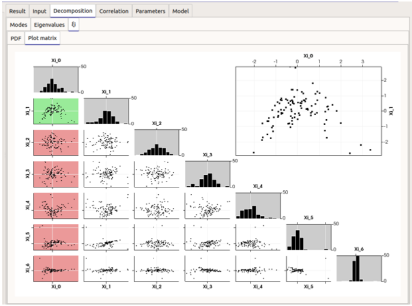
To analyze these trajectories requires more advanced tools than with classical multivariate samples, so that we can take into account for the time dependence.

We use the BagChart and Functional Bagchart (from Paraview) tools, which uses the High Density Regions algorithm (Hyndman, 1996).

PERSALYS: 1D fields

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

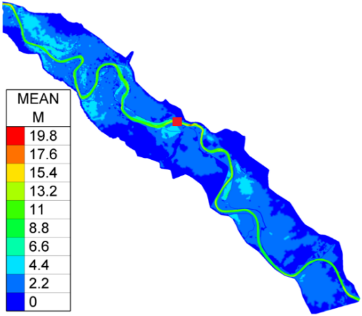




What's next ?

PERSALYS Roadmap :

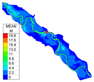
- Calibration
- 2D Fields, 3D Fields
- In-Situ fields based on the MELISSA library (with INRIA)



What's next ?

PERSALYS Roadmap :

- Calibration
- 2D Fields, 3D Fields
- In-Situ fields based on the MELISSA library (with INRIA)



We are currently working on adding features to perform the calibration of a computer code (almost done).
The next features we plan to add to PERSALYS are the management of 2D and 3D stochastic fields.
We also work on the use of the MELISSA software which performs in-situ studies.

This library allows to perform UQ studies in situations where we cannot store more than a couple of multidimensional fields in memory or on the hard drive.

The end

Thanks !

Questions ?

Thanks !

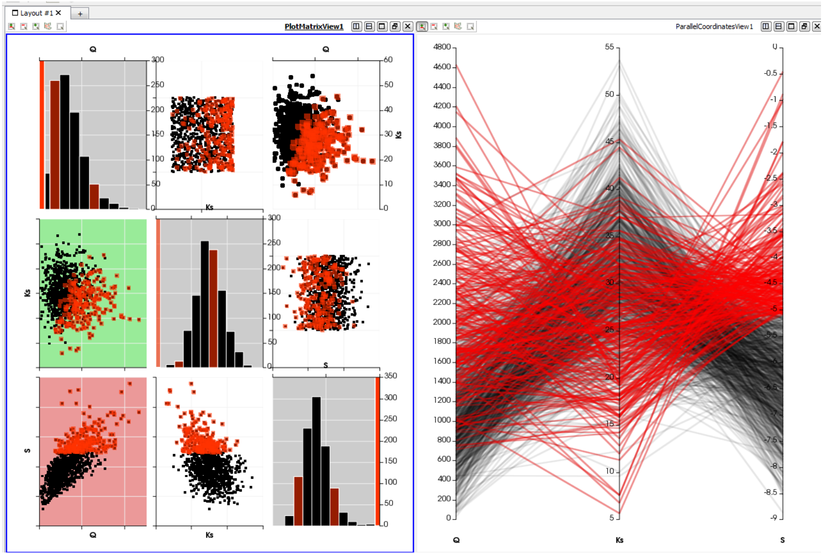
Questions ?

Thank you for your attention.

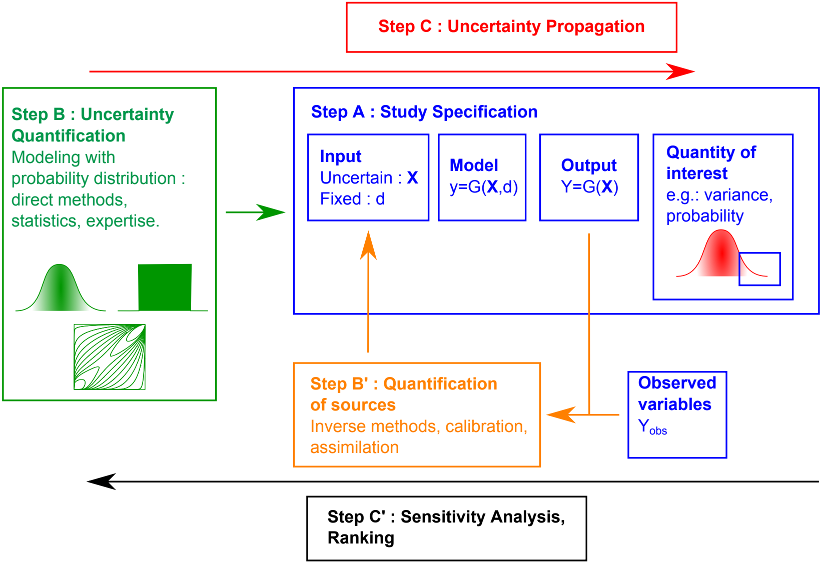
If you have any question, it would be a pleasure to answer them.

If you want a live demo of PERSALYS, I can show you during the coffee break.

Interactive uncertainty visualization with Paraview



Methodology



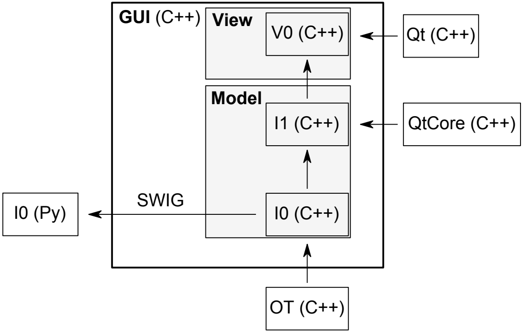
Software architecture

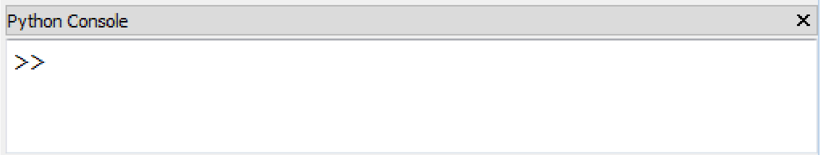
Two entry points:

- interactive,
- Python.

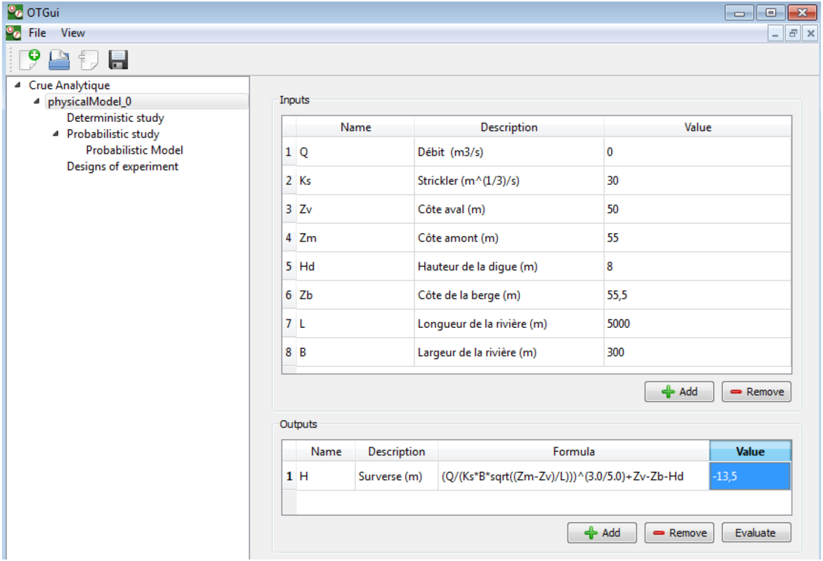
Advantages of the Python programming of the GUI:

- unit tests,
- going beyond the GUI

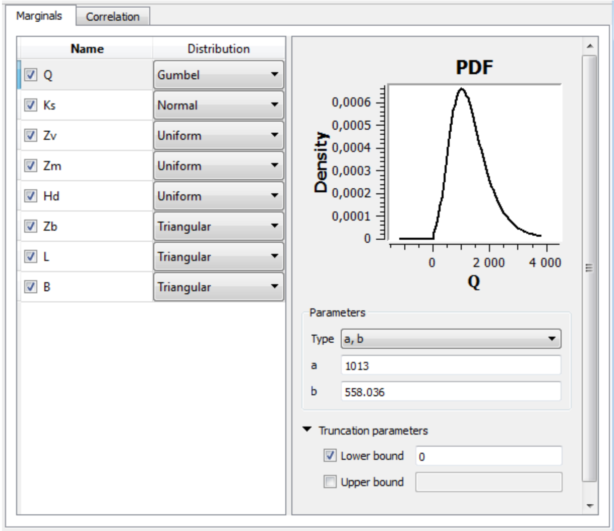




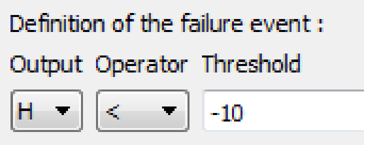
Symbolic physical model



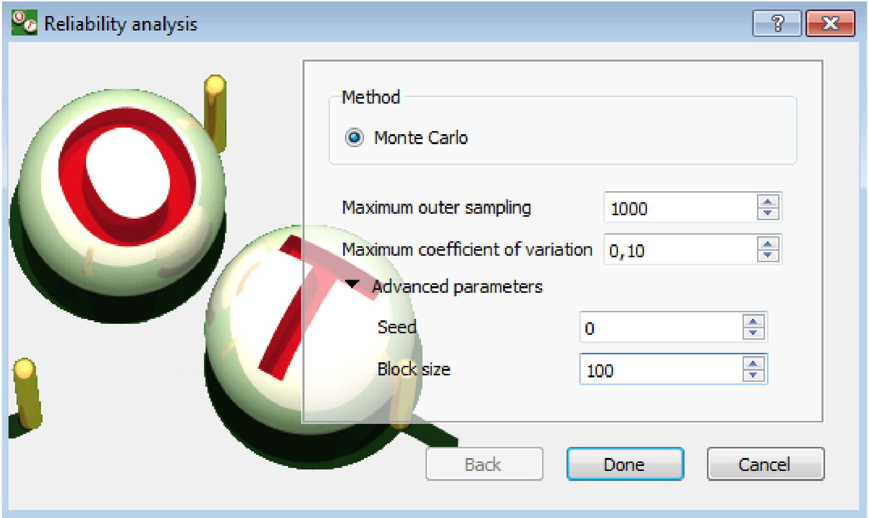
Probabilistic model



Limit state study : definition of the threshold



Limit state study : algorithm parameters



Limit state study : summary

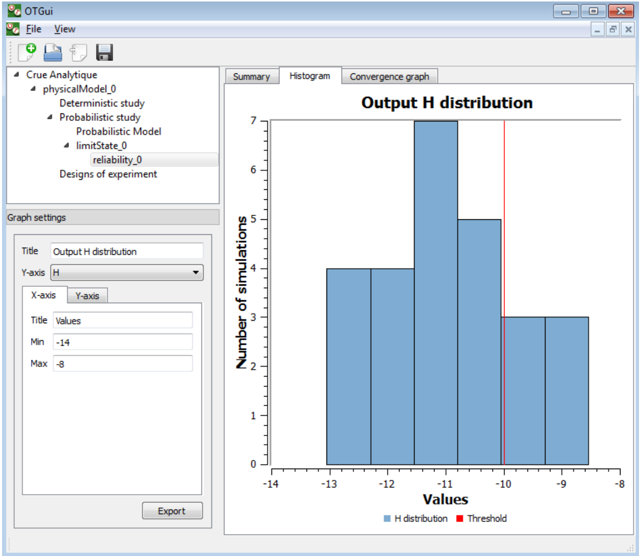
Summary | Histogram | Convergence graph

Output H

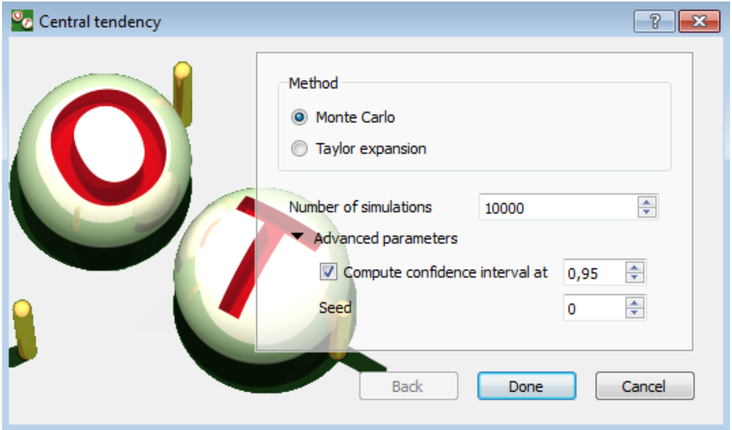
Number of simulations: 26

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

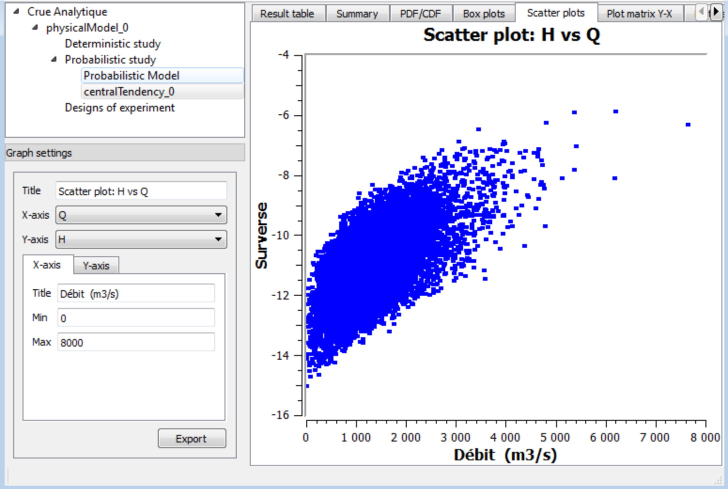
Output H

Number of simulations: 10000

Minimum and Maximum

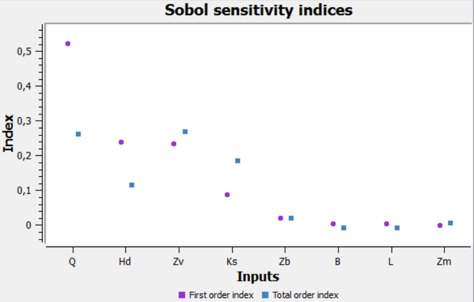
	Variable	Minimum	Maximum
Output	H	-15.0155	-5.88758
Inputs at extremum	Q	7.97827	6187.43
	Ks	27.132	25.5926
	Zv	49.1681	50.9071
	Zm	54.5469	55.3994
	Hd	8.76082	8.49391
	Zb	55.5436	55.4935
	L	4999.26	4997.37
	B	303.187	300.871

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 ⚠

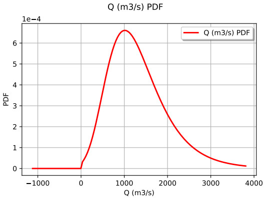
OpenTURNS: estimate the mean

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