Overview of OpenTURNS, its new features and its graphical user interface

M. Baudin [1] T. Delage [1] A. Dutfoy [1] A. Geay [1]
O. Mircescu [1] A. Ladier [2] J. Schueller [2] T. 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

27 March 2020, SIAM UQ 2020, Munich, Germany

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
- 5 full time developers
- Users $\approx$ 1000, mainly in France (370 000 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

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

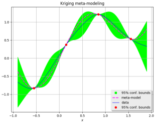
Sampling methods: Monte Carlo, LHS, low discrepancy sequences
Variance reduction methods: importance sampling, subset sampling
Approximation methods: FORM, SORM
Indices: Spearman, Sobol, ANCOVA
Importance factors: perturbation method, FORM, Monte Carlo

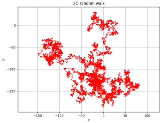
Functional modeling

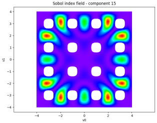
Numerical functions: symbolic, Python-defined, user-defined
Function operators: addition, product, composition, gradients
Function transformation: linear combination, aggregation, parametrization
Polynomials: orthogonal polynomial, algebra

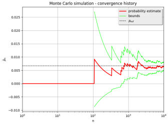
Numerical methods

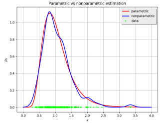
Integration: Gauss-Kronrod
Optimization: NLopt, Cobyla, TNC
Root finding: Brent, Bisection
Linear algrebra: Matrix, HMat
Interpolation: piecewise linear, piecewise Hermite
Least squares: SVD, QR, Cholesky







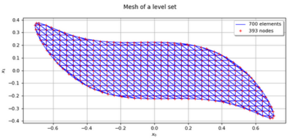




OpenTURNS: documentation

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 ≤ 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.

- 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:

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

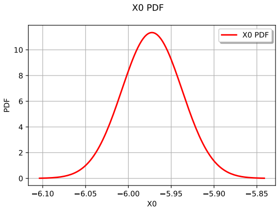
- The absolute accuracy of the estimate $\bar{\mu}$ can be evaluated based on the sample standard deviation of the estimator $\hat{s}/\sqrt{n}$
- To get good performances on distributed supercomputers and multi-core workstations, the size of the sample increases by block.

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:

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

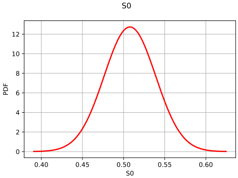
  where Ψ is a multivariate function, U is a multivariate sample and $\overline{U}$ is its sample mean.
- Each Sobol' estimator (e.g. Saltelli, Jansen, etc...) can be associated with a specific choice of function Ψ and vector U.
- Therefore, the multivariate delta method implies:

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

  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 (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()
```



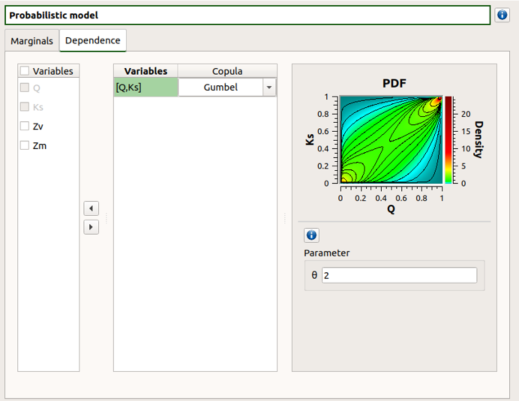
Asymptotic distribution of the first order Sobol' indices for the first variable.

PERSALYS, the graphical user interface of OpenTURNS

- Provide a graphical interface of OpenTURNS in and out of the SALOME integration platform
- Features : probabilistic model, distribution fitting, central tendency, sensitivity analysis, probability estimate, meta-modeling (polynomial chaos, kriging), screening (Morris), optimization, design of experiments
- GUI language : English, French
- Partners : EDF, Phiméca
- Licence : LGPL
- Schedule : Since summer 2016, two EDF release per year
- On the internet (free) : SALOME_EDF since 2018 on `www.salome-platform.org`

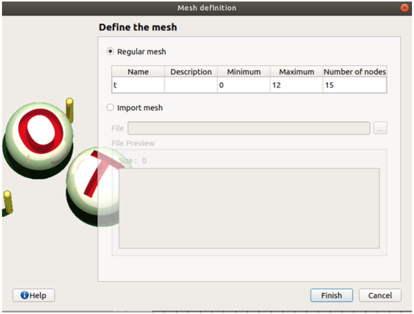
PERSALYS: define the dependence

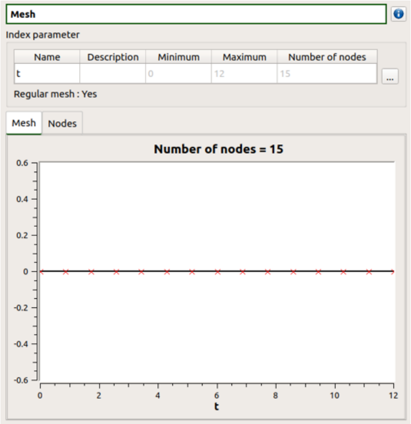
- Dependence is defined using copulas
- Define arbitrary groups of dependent variables
- Available copulas (same as in OT): gaussian, Ali-Mikhail-Haq, Clayton, Farlie-Gumbel-Morgenstern, Frank, Gumbel
- Dependence inference from a sample : Bayesian Information Criteria (BIC) or Kendall plot



PERSALYS: 1D fields

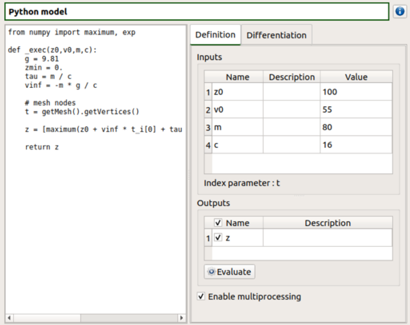
- Mesh definition and visualization
- Import from text or csv file

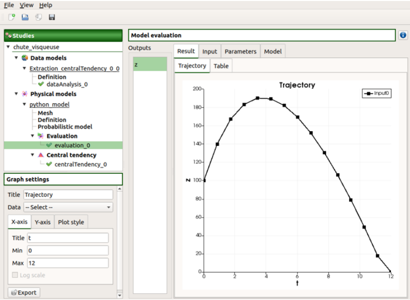




PERSALYS: 1D fields

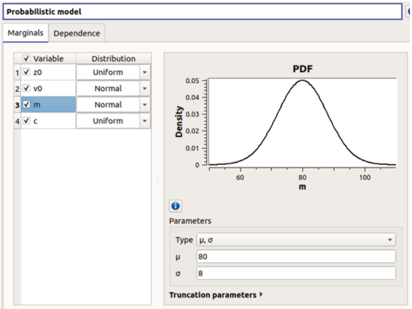
- Functional model definition and probabilistic model
- Python or symbolic

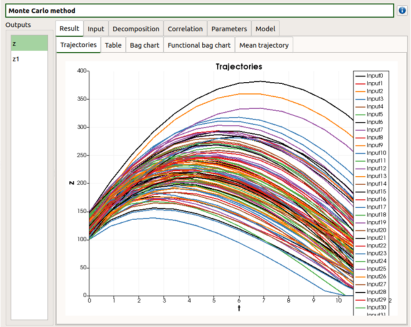




PERSALYS: 1D fields

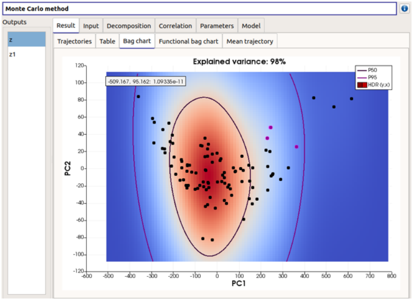
- Probabilistic model
- Uncertainty propagation with simple Monte-Carlo sampling

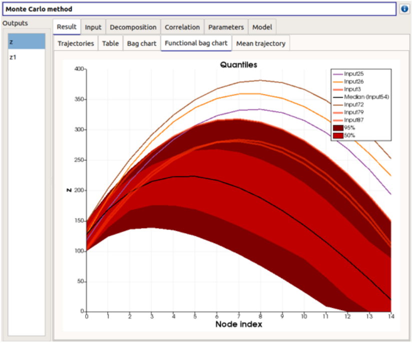




PERSALYS: 1D fields

- 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.

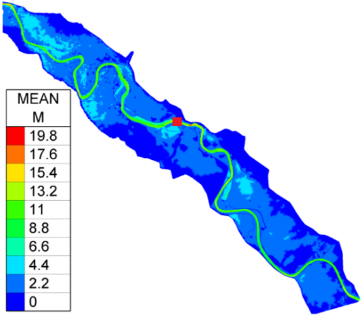




What's next ?

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 ?