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

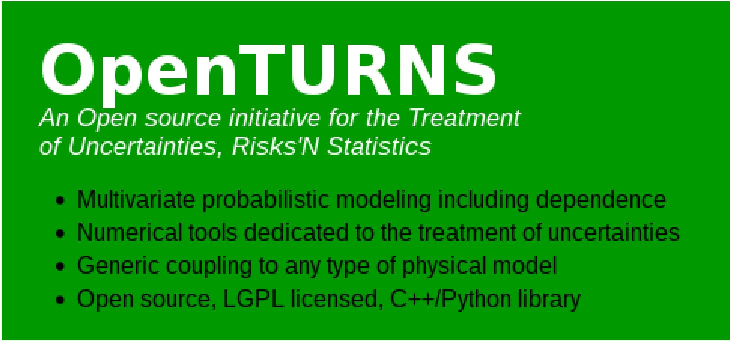
J. Muré [1] M. Baudin [1] J. Pelamatti [1] A. Dutfoy [1]
O. Mircescu [1] J. Schueller [2] T. Yalamas [2]

[1]EDF R&D. 6, quai Watier, 78401, Chatou Cedex - France, joseph.mure@edf.fr

[2]Phimeca Engineering. 18/20 boulevard de Reuilly, 75012 Paris - France, yalamas@phimeca.com

April 15th 2022, SIAM UQ 2022, Atlanta, US

OpenTURNS: `www.openturns.org`



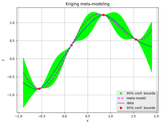
- Multivariate probabilistic modeling including dependence
- Numerical tools dedicated to the treatment of uncertainties
- Generic coupling to any type of physical model
- Open source, LGPL licensed, C++/Python library

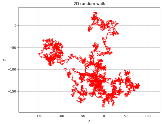
OpenTURNS: www.openturns.org

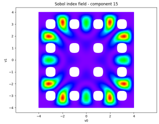
- Linux, Windows, macOS
- First release : 2007
- 5 full time developers
- Users ≈ 1000, mainly in France (785 000 Total Conda downloads)
- Project size : 800 classes, more than 6000 services

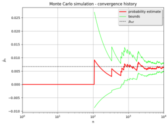
OpenTURNS: content

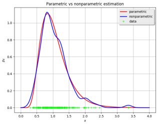
- Data analysis
  - Distribution fitting
  - Statistical tests
  - Estimate dependency and copulas
  - Estimate stochastic processes
- Probabilistic modeling
  - Dependence modeling
  - Univariate distributions
  - Multivariate distrbutions
  - Copulas
  - Processes
  - Covariance kernels
- Surrogate models
  - Linear regression
  - Polynomial chaos expansion
  - Gaussian process regression
  - Spectral methods
  - Low rank tensors
  - Fields metamodel
- Reliability, sensitivity
  - Sampling methods
  - Approximation methods
  - Sensitivity analysis
  - Design of experiments
- Calibration
  - Least squares calibration
  - Gaussian calibration
  - Bayesian calibration
- Numerical methods
  - Optimization
  - Integration
  - Least squares
  - Meshing
  - Coupling with external codes







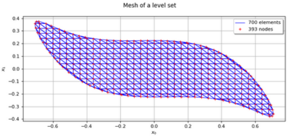




OpenTURNS: documentation

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.

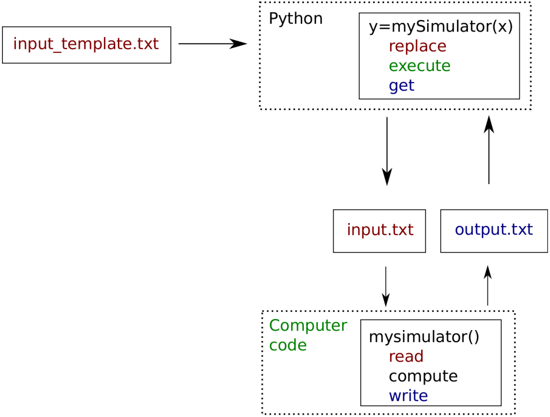
- Content:
  - Programming interface (API)
  - Examples
  - Theory
- *All* classes and methods are documented, partly automatically.
- Examples are automatically tested at *each* update of the code and outputs are checked.

OpenTURNS: practical use

- Compatibility with most popular python packages
  - Numpy
  - Scipy
  - Matplotlib
  - scikit-learn
- Parallel computational with shared memory (TBB)
- Optimized linear algebra with LAPACK and BLAS
- Possibility to interface with a computation cluster
- Focused towards handling numerical data
- Installation through conda, pip, packages for various Linux distros and source code

Coupling OpenTURNS with computer codes

OpenTURNS provides a text file exchange based interface in order to perform analyses on complex computer codes



- Replaces the need for input/output text parsers
- Wraps a simulation code under the form of a standard python function
- Allows to interface OpenTURNS with a cluster
- otwrapy: interfacing tool to allow easy parallelization

OpenTURNS Sensitivity analysis : HSIC indices[1][2]

- Probabilistic modeling of d input physical variables and black-box model:

$$\mathbf{X} = (X_1, X_2, \ldots, X_d)^\top \sim P_{\mathbf{X}} \quad \text{over} \quad \mathcal{X} = \times_{i=1}^{d} \mathcal{X}_i$$

$$\mathcal{M} : \left| \begin{array}{ccc} \mathcal{X} \subseteq \mathbb{R}^d & \longrightarrow & \mathcal{Y} \subseteq \mathbb{R} \\ \mathbf{X} & \longrightarrow & Y = \mathcal{M}(\mathbf{X}) \end{array} \right.$$

- **Learning sample** $\rightarrow$ a n-size i.i.d. sample of the couple $(\mathbf{X}, Y)$:

$$\left(\mathbf{X}^{(j)}, Y^{(j)}\right)_{(1 \leq j \leq n)} = \left(X_1^{(j)}, X_2^{(j)}, \ldots, X_d^{(j)}, Y^{(j)}\right)_{(1 \leq j \leq n)}$$

with $P_{\mathbf{X}^{(j)}} = P_{\mathbf{X}}$ and $Y^{(j)} = \mathcal{M}\left(X_1^{(j)}, X_2^{(j)}, \ldots, X_d^{(j)}\right), \forall j \in \{1, \ldots, n\}$

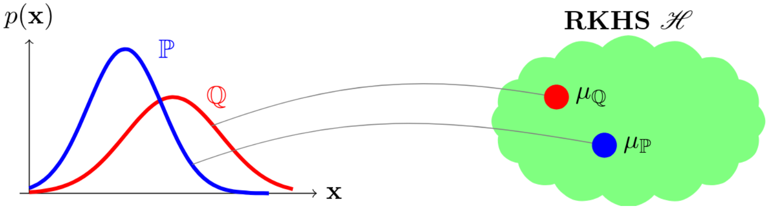
[1] **gretton2005**.

[2] **daveiga2015**.

OpenTURNS Sensitivity analysis : HSIC indices

- Let $\mathcal{V}_i$ be a Reproducing Kernel Hibert Space over $\mathcal{X}_i \times \mathcal{Y}$ with kernel $v_i(\cdot, \cdot)$.
- We consider the mean embedding of $P_{X_i,Y}$ and $P_{X_i}P_Y$ in $\mathcal{V}$:
  - $\mu[P_{X_i,Y}] = \iint_{\mathcal{X}_i \times \mathcal{Y}} v_i((x,y),\cdot)dP_{X_i,Y}(x,y)$
  - $\mu[P_{X_i}P_Y] = \int_{\mathcal{Y}} \left[\int_{\mathcal{X}_i} v_i((x,y),\cdot)dP_{X_i}(x)\right] dP_Y(y)$
- We now have a dependence measure under the form of:

$$\Delta := ||\mu[P_{Y,X_i}] - \mu[P_Y P_{X_i}]|| \longleftrightarrow HSIC(Y, X_i) = \Delta^2$$



OpenTURNS Sensitivity analysis : HSIC indices

A few advantages

- Data efficient
- Given data estimators
- Computational cost scales linearly with the problem dimension
- Associated statistical test
- Can be used when dealing with non continuous variables
  - discrete/categorical variables
  - graphs
  - stochastic codes
- Formulation holds in case of dependence between inputs
  - Interpretation of results becomes complicated!

OpenTURNS Sensitivity analysis : HSIC indices

Statistical hypothesis testing with HSIC

- We consider the following statistical test $\mathcal{T}$

$$\mathcal{T} : \text{Test} \quad \text{"}(\mathcal{H}_{0,i}) : \text{HSIC}(X_i, Y) = 0\text{"} \quad \text{vs.} \quad \text{"}(\mathcal{H}_{1,i}) : \text{HSIC}(X_i, Y) > 0\text{"}$$

$\mathcal{H}_{0,i}$: X_i and Y are independent

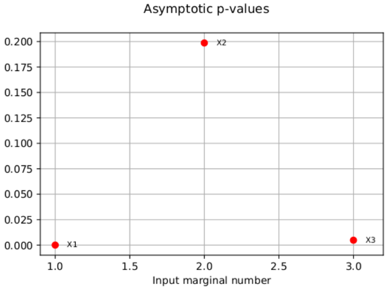
- $\widehat{S}_{\mathcal{T}} := n \times \widehat{\text{HSIC}}(X_i, Y)$ is the test statistic, $\widehat{S}_{\mathcal{T},\text{obs}}$ its observed value.
- **p-value** associated to the test $\mathcal{T}$:

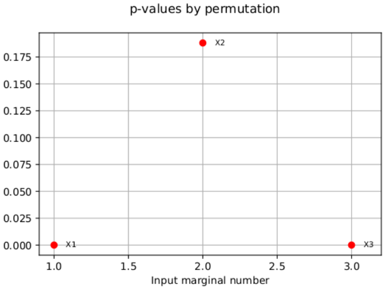
$$p_{\text{val}} = \mathbb{P}\left(\widehat{S}_{\mathcal{T}} \geq \widehat{S}_{\mathcal{T},\text{obs}} \mid \mathcal{H}_{0,i}\right)$$

  - Asymptotic estimator (large data set)
  - Permutation-based estimator (small data set)

OpenTURNS Sensitivity analysis : HSIC indices

```
globHSIC = ot.HSICEstimatorGlobalSensitivity(X, Y, kernel_list, ot.HSICUStat())
globHSIC.drawPValuesAsymptotic()
globHSIC.drawPValuesPermutation()
```





HSIC indices are compatible with various sensitivity analysis objectives:

- Global analysis
- Target analysis, e.g. sensitivity of the event $\{Y > s\}$ to the X_i ($s \in \mathbb{R}$)
- Conditional analysis, e.g. sensitivity of Y to the X_i conditionally to $Y > s$

OpenTURNS: Metropolis-Hastings

We want to sample from the joint distribution π of 3 random variables X, Y, Z. Here is one step of the algorithm, starting from the point (x, y, z):

1. Simulate a candidate $(x', y', z') \sim q(\cdot|x, y, z)$ for some conditional distribution q.
2. Compute $\alpha(x', y', z'|x, y, z) = \min\left\{ \frac{\pi(x',y',z')\, q(x,y,z|x',y',z')}{\pi(x,y,z) q(x',y',z'|x,y,z)}, 1 \right\}$.
3. Simulate $u \sim \mathcal{U}(0, 1)$. If $u \leqslant \alpha(x', y', z'|x, y, z)$, then the next state is (x', y', z'), otherwise it is (x, y, z).

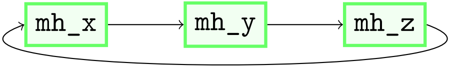
Random walk Metropolis-Hastings

When $q(\cdot|x, y, z) = (x, y, z) + \mu$, where μ is a distribution that does not depend on (x, y, z), the algorithm is called "Random walk Metropolis-Hastings" and μ is called the "proposal distribution".

```
init = [x_start, y_start, z_start]
proposal = ot.Normal(3) # steps follow a 3d normal distribution
mh = ot.RandomWalkMetropolisHastings(logdensity, support, init, proposal)
sample = mh.getSample(1000)
```

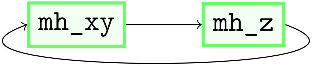
OpenTURNS: Metropolis-Hastings variants

Single component randomwalk Metropolis-Hastings




```
prop_dim1 = ot.Normal(1) # steps follow a 1d normal distribution along one axis
mh_x = ot.RandomWalkMetropolisHastings(logdensity, support, init, prop_dim1, [0])
mh_y = ot.RandomWalkMetropolisHastings(logdensity, support, init, prop_dim1, [1])
mh_z = ot.RandomWalkMetropolisHastings(logdensity, support, init, prop_dim1, [2])
sample = ot.Gibbs([mh_x, mh_y, mh_z]).getSample(1000)
```

Blocks of components can be considered




```
prop_xy = ot.Normal(2) # steps follows a 2d normal distribution for (X,Y)
prop_z = ot.Normal(1)  # steps follow  a 1d normal distribution for Z
mh_xy = ot.RandomWalkMetropolisHastings(logdensity, support, init, prop_xy, [0, 1])
mh_z = ot.RandomWalkMetropolisHastings(logdensity, support, init, proposal, [2])
sample = ot.Gibbs([mh_xy, mh_z]).getSample(1000)
```

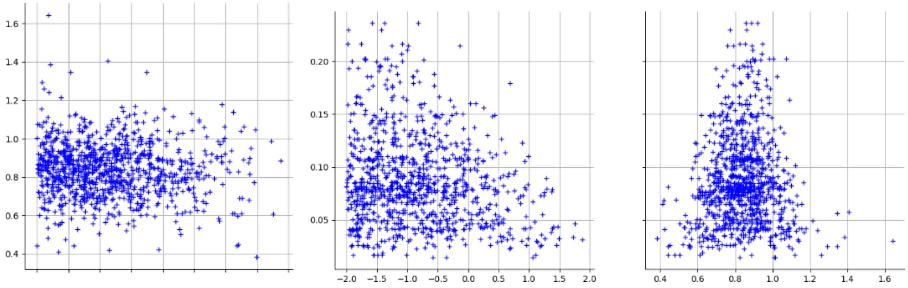
OpenTURNS: conditional samplers in Gibbs algorithms

Assume we have a direct sampler for Z conditionally on X and Y: this sampler should be passed to the `RandomVectorMetropolisHastings` class.

```
z_sampler = ot.RandomVector(...) # build a direct sampler for Z | X, Y
# Typically, Z depends on a function of (X,Y)
# Write it as a function of (X,Y,Z) for the sake of genericity:
f = ot.Function(...) # function f(X,Y,Z) which outputs the parameters of z_sampler
mh_z = ot.RandomVectorMetropolisHastings(z_sampler, init, [2], f)

prop_xy = ot.Normal(2) # steps follows a 2d normal distribution for (X,Y)
mh_xy = ot.RandomWalkMetropolisHastings(logdensity, support, init, prop_xy, [0, 1])

sample = ot.Gibbs([mh_x, mh_y, mh_z]).getSample(1000)
```



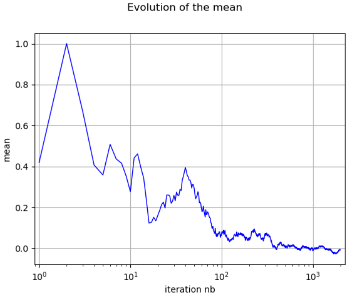
OpenTURNS: iterative statistics[3][4]

Iterative moment estimation

```
iterMoments = ot.IterativeMoments(order, dim)

# Iteratively compute the mean
size = 2000
meanEvolution = ot.Sample()
for i in range(size):
    point = distNormal.getRealization()
    iterMoments.increment(point)
    meanEvolution.add(iterMoments.getMean())

# Display the evolution
ran = range(1, size + 1)
iter_sample = ot.Sample.BuildFromPoint(ran)
curve = ot.Curve(iter_sample, meanEvolution)
graph = ot.Graph("Evolution of the mean",
                 "iteration nb",
                 "mean",
                 True)
graph.add(curve)
graph.setLogScale(ot.GraphImplementation.LOGX)
view = otv.View(graph)
```



Other available iterative stats

- Extrema
- Threshold exceedance probability

[3] **pebay2008**.

[4] **meng2015**.

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, surrogate modeling (polynomial chaos, kriging), screening (Morris), optimization, design of experiments
- GUI language : English, French
- Partners : EDF, Phiméca
- Licence : LGPL
- Schedule : Since summer 2016, two releases per year, currently V12
- On the internet (free) : SALOME_EDF since 2018 on `www.salome-platform.org`

Persalys

Calibration

Given a physical model h, observed inputs $\boldsymbol{x}$, observed outputs y we can calibrate $\boldsymbol{\theta}$ so that

$$y = h(\boldsymbol{x}, \boldsymbol{\theta}) + \epsilon$$

where ϵ is a random (Gaussian) variable.
Calibration outputs:

- the optimal value estimator $\hat{\boldsymbol{\theta}}$,
- the distribution of $\hat{\boldsymbol{\theta}}$ and a confidence or credibility interval of each marginal,
- the distribution of the residuals $\epsilon = y - h(\boldsymbol{x}, \hat{\boldsymbol{\theta}})$.

We implemented 4 methods[5] :

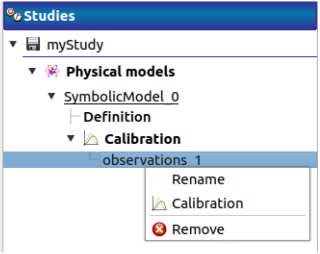
	Linear	**Non Linear**
No prior	Least squares linear	Least squares non linear
Gaussian prior	Linear calib.	3DVAR

[5] M. Baudin & R. Lebrun, Linear algebra of linear and nonlinear Bayesian calibration. UNCECOMP 2021.

Calibration

In PERSALYS, this requires:

- a physical model h with parameters to calibrate,
- a data file containing the observed inputs and outputs.

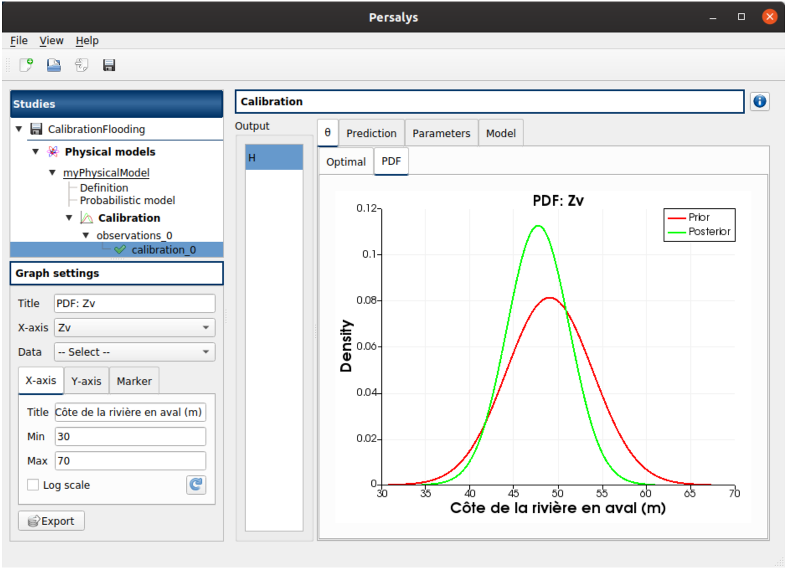


On output, we present the optimal value of each calibrated parameter, along with a 95% confidence interval.

Optimal θ

Input	Value	Confidence interval at 95%
Ks	20.0432	[15.5523, 24.5341]
Zv	47.7043	[39.7458, 55.6628]
Zm	52.4036	[44.4431, 60.3641]

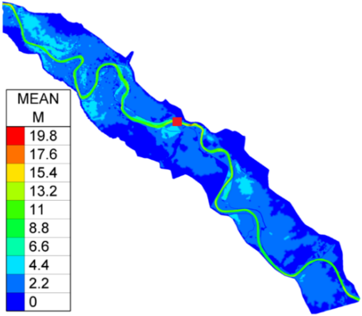
Calibration



What's next ?

PERSALYS Roadmap :

- New surrogate models
- 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 ?