OpenTURNS & **Persalys** overview: two open source tools for Uncertainty Quantification and Data Analysis

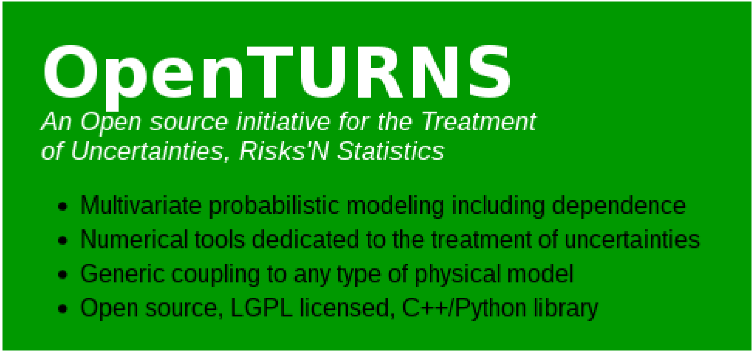
The OpenTURNS & Persalys Dev Team

Prepared by Julien Pelamatti[1], Vincent Chabridon[1] and Elias Fekhari[1]

[1]EDF R&D, 6 Quai Watier, 78401 Chatou, France

HIPERWIND Summer School, August 28 – 31, 2023,
DTU Risø Campus, Roskilde, Denmark

OpenTURNS: www.openturns.org



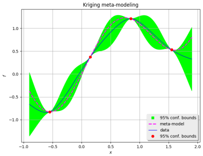
- Multivariate probabilistic modeling including complex dependencies
- Numerical tools dedicated to the treatment of uncertainties
- Generic coupling to any type of physical model
- **Open source, LGPL licensed, C++ core with a Python API**

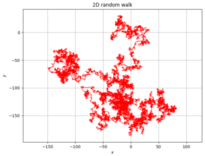
OpenTURNS: www.openturns.org

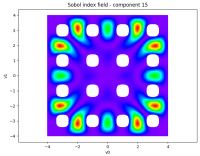
- Supported operating systems: Linux, Windows, macOS, Android
- First release: 2007
- 5 **full time developers** / $\sim$ 10 **regular contributors**
- $\sim$ **1 000 000 total Conda downloads since 2016,** $\sim$ **5000 monthly Pipy downloads**
- Project size: 800 classes, more than 6000 services
- Available in Debian *Bookworm*, Ubuntu, ...

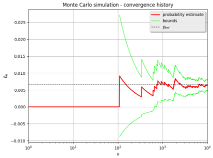
OpenTURNS: content

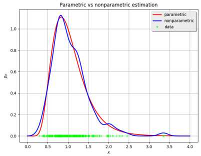
- Data analysis
  - Parametric Distribution fitting
  - Non-parametric Distribution fitting
  - Statistical tests
  - Estimate dependency and copulas
  - Estimate stochastic processes
- Probabilistic modeling
  - Dependence modeling
  - Univariate distributions
  - Multivariate distributions
  - Copulas
  - 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







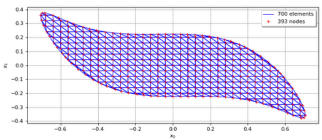




OpenTURNS: documentation

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 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 level set type.
`getClassName`()	Accessor to the object's name.
`getDiscretization`()	Accessor to the discretization.
`getId`()	Accessor to the object's id.
`getName`()	Accessor to the object's name.
`getOptimizationAlgorithm`()	Accessor to the optimization solver.
`getShadowedId`()	Accessor to the object's shadowed id.
`getVisibility`()	Accessor to the object's visibility state.
`hasName`()	Test if the object is named.
`hasVisibleName`()	Test if the object has a distinguishable name.
`setDiscretization`(discretization)	Accessor to the discretization.
`setName`(name)	Accessor to the object's name.
`setOptimizationAlgorithm`(solver)	Accessor to the optimization solver.
`setShadowedId`(id)	Accessor to the object's shadowed id.
`setVisibility`(visible)	Accessor to the object's visibility state.

__init__(**args*)

build(**args*)

Build the mesh of level set type.

Parameters: **levelSet** : `LevelSet`

The level set to be meshed, of dimension equal to the dimension of `discretization`.

boundingBox : `Interval`

The bounding box used to mesh the level set.

project : *bool*

Flag to tell if the vertices outside of the level set of a simplex partially included into the level set have to be projected onto the level set. Default is *True*.

Returns: **mesh** : `Mesh`

The mesh built.

- Content:
  - Programming interface (API)
  - Examples
  - Theory
- *All* classes and methods are documented, partly automatically.
- Examples and unit tests are automatically run at *each* code update

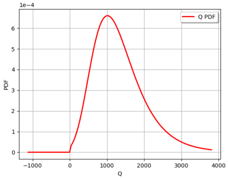
OpenTURNS: practical use

- Compatibility with several popular python packages
  - Numpy
  - Scipy
  - Matplotlib
  - Scikit-learn
  - Pandas
- 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

Probabilistic modeling

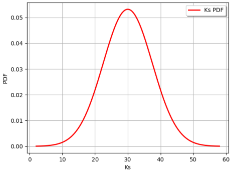
Random variables distributions:

Q: Gumbel(scale=558, mode=1013)>0



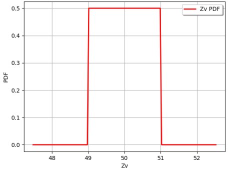

```
Dist = ot.Gumbel(558,1013)
Q = ot.TruncatedDistribution(Dist, 0.,
ot.TruncatedDistribution.LOWER)
```

Ks: Normal(mean=30, std=7.5)>0

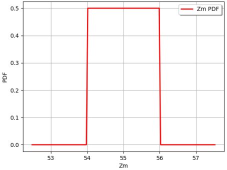



```
Dist = ot.Normal(30.,7.5)
Ks = ot.TruncatedDistribution(Dist, 0., ot.
    TruncatedDistribution.LOWER)
```

Zv: Uniform(min=49, max=51)



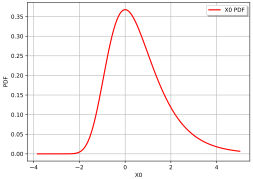
Zm: Uniform(min=54, max=56)

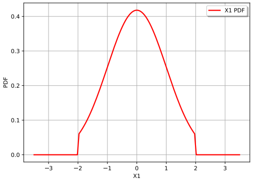


Joint probability distributions

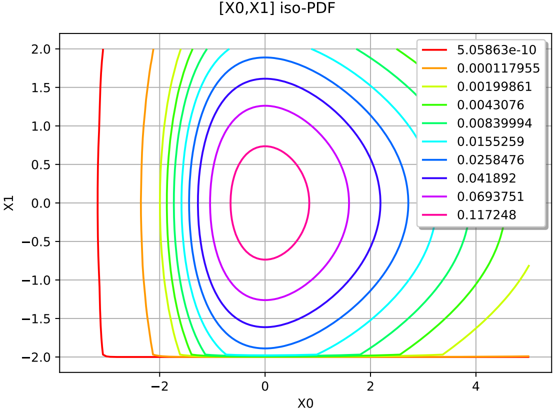
- We consider a 2-dimensional distribution with the following marginals:
  - Gumbel(min = -1, max = 1)
  - Truncated normal (mean = 0, std = 1, min = -2, max = 2)

Marginals

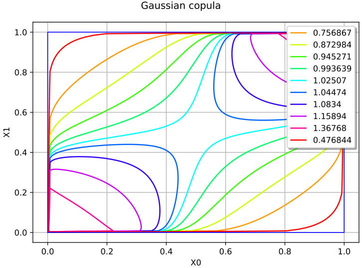


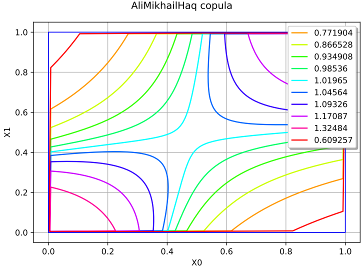


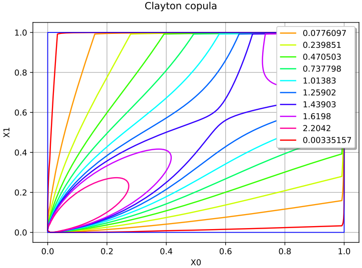
Joint distributions

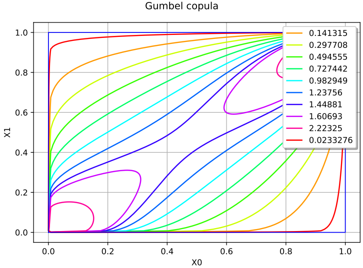


Beyond independent marginals: Copulas

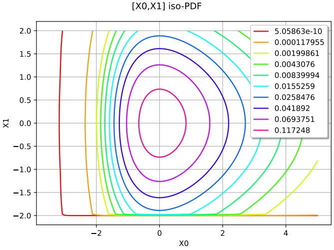


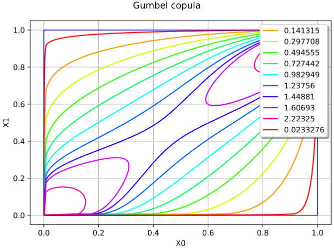




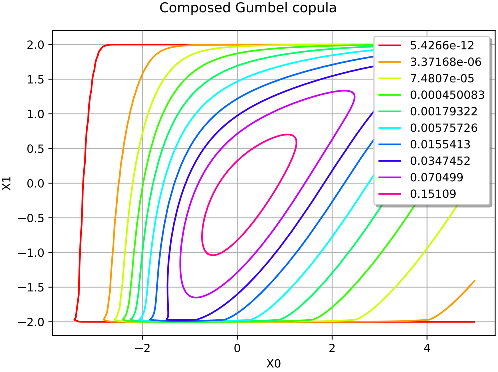


Composing marginal distributions and copulas



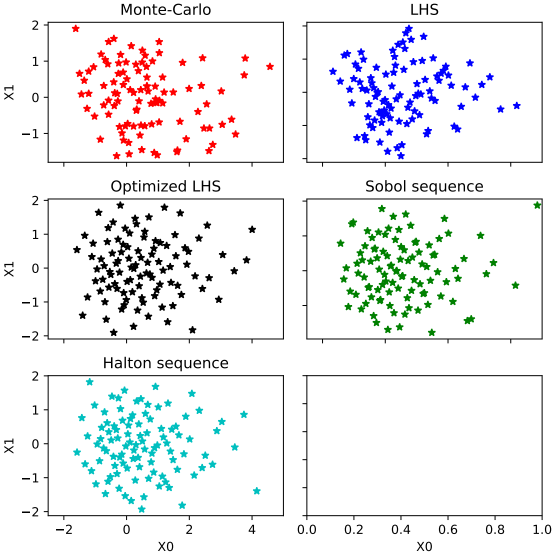


<u>We obtain:</u>




```
distribution =
 [ot.Uniform(),ot.TruncatedNormal(0,1,-2,2)]
composed = ot.ComposedDistribution(X,copula)
graph = composed.drawPDF()
graph.setTitle('Composed Gumbel copula')
viewer.View(graph)
```

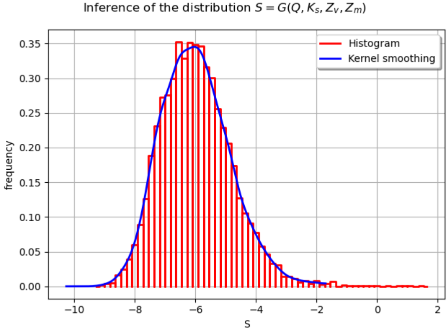
Design of experiments




```python
dim = 2
X = [ot.Gumbel(),ot.TruncatedNormal(0,1,-2,2)]
distribution = ot.ComposedDistribution(X)
bounds = distribution.getRange()
sampleSize = 100

sample1 = distribution.getSample(sampleSize)

experiment = ot.LHSExperiment(distribution,
  sampleSize, False, False)
sample2 = experiment.generate()

lhs = ot.LHSExperiment(distribution,
 sampleSize)
lhs.setAlwaysShuffle(True) # randomized
space_filling = ot.SpaceFillingC2()
temperatureProfile = ot.GeometricProfile(10.0, 0.95,
     1000)
algo = ot.SimulatedAnnealingLHS(lhs,
 space_filling, temperatureProfile)
sample3 = algo.generate()

sequence = ot.SobolSequence(dim)
experiment = ot.LowDiscrepancyExperiment(
  sequence, distribution, sampleSize, False)
sample4 = experiment.generate()
```

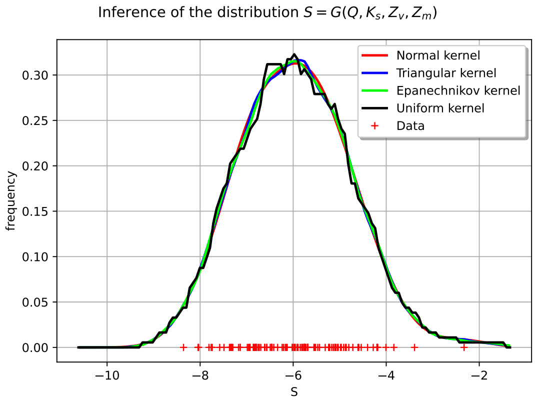
Monte Carlo sampling

- The input distribution and relative output value are evaluated 10000 times
- The output distribution can be infered as a parametric function or through histogram or kernel smoothing methods




```
Distribution = ot.ComposedDistribution([Q,Ks,Zv,Zm])

#Python model
def floodFunction(X):
    Q, Ks, Zv, Zm = X
    alpha = (Zm - Zv)/5.0e3
    H = (Q/(300.0*Ks*np.sqrt(alpha)))**0.6
    S = [H + Zv - 58.5]
    return S

fun = ot.PythonFunction(4,1,floodFunction)

#We define the output as a random vector
inputVector = ot.RandomVector(Distribution)
outputVector = ot.CompositeRandomVector(fun,
 inputVector)

#We sample and infere the output distribution
size = 10000
sampleY = outputVector.getSample(size)
graph = ot.HistogramFactory().build(sampleY).drawPDF()
loiKS = ot.KernelSmoothing().build(sampleY)
graph2 = loiKS.drawPDF()
```

Distribution and dependence inference

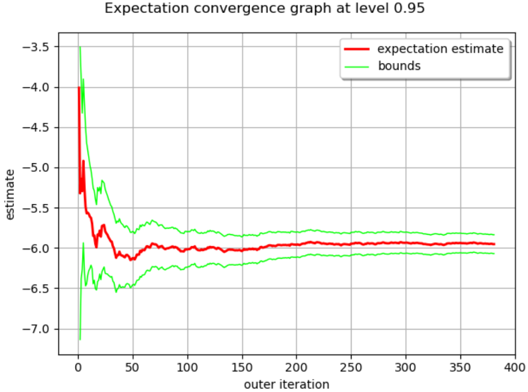



```
size = 100
sampleY = outputVector.getSample(size)
graph = ot.KernelSmoothing(ot.Normal()).build
     (sampleY).drawPDF()
loiKS = ot.KernelSmoothing(ot.Triangular()).
     build(sampleY)
graph2 = loiKS.drawPDF()
graph.add(graph2)
loiKS = ot.KernelSmoothing(ot.Epanechnikov())
     .build(sampleY)
graph2 = loiKS.drawPDF()
graph.add(graph2)
loiKS = ot.KernelSmoothing(ot.Uniform()).
     build(sampleY)
graph2 = loiKS.drawPDF()
```

- Parametric ($1d - Nd$) distribution inference
- Non-parametric ($1d - Nd$) distribution inference
- Parametric copula inference
- Non-parametric copula inference (Bernstein copula)
- Resampling w.r.t. inferred distributions

Iterative Monte Carlo: Central tendency anaysis

- The expected value and associated standard deviation are computed iteratively
- Different stopping criteria can be used
- Batch computation can be used



$$\widehat{m_y} = \frac{1}{N}\sum_{1}^{N} G(\mathbf{X}_i)$$

$$\widehat{\sigma_y} = \sqrt{\frac{1}{N-1}\sum_{1}^{N}(G(\mathbf{X}_i) - \hat{m}_y)^2}$$

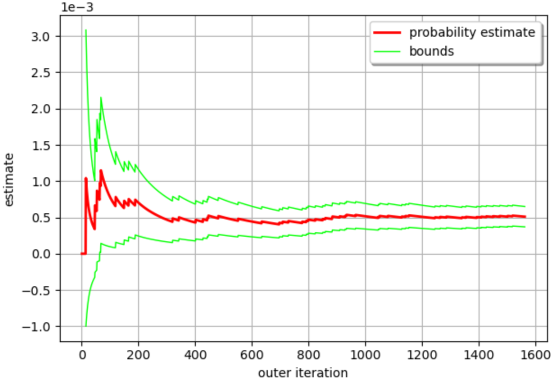
$$\widehat{\sigma_{m_y}} = \widehat{\sigma_y}/\sqrt{N}$$

```
algo = ot.ExpectationSimulationAlgorithm(
 outputVector)
algo.setMaximumOuterSampling(100000)
algo.setBlockSize(1)
algo.setCoefficientOfVariationCriterionType(
 'MAX')
algo.setMaximumCoefficientOfVariation(0.01)
algo.run()
graph = algo.drawExpectationConvergence()
view = View(graph)
```

Iterative Monte Carlo: Reliability analysis

- We now consider the probability of flooding: $(P(S > 0))$
- Same as before, but the function $\mathbb{I}_{G(\mathbf{X}_i)>0}$ is considered

ProbabilitySimulationAlgorithm convergence graph at level 0.95



$$\widehat{p} = \frac{1}{N}\sum_1^N \mathbb{I}_{G(\mathbf{X}_i)>0}$$

$$\widehat{\sigma} = \sqrt{\frac{1}{N-1}\sum_1^N \left(\mathbb{I}_{G(\mathbf{X}_i)>0} - \widehat{p}\right)^2}$$

$$\widehat{\sigma_p} = \widehat{\sigma}/\sqrt{N}$$

```
eventF = ot.ThresholdEvent(outputVector, ot.
     GreaterOrEqual(), 0.0)
exp = ot.MonteCarloExperiment()
algo = ot.ProbabilitySimulationAlgorithm(eventF, exp)
algo.setMaximumOuterSampling(100000)
algo.setMaximumCoefficientOfVariation(0.01)
algo.setBlockSize(10)
algo.run()
```

FORM/SORM reliability analysis

- We estimate the probability of flooding through FORM/SORM procedures
- MC estimation requires $\simeq$ 1500 function evaluations
- FORM and SORM only use $\simeq$ 150
- Estimated probability:
  - MC: 5.09999999999998 1e-4
  - FORM: 5.340929030055227 1e-4
  - SORM: 6.793780433482759 1e-4

```
#FORM
OptAlgo = ot.Cobyla()
startingPoint = Distribution.getMean()
algoFORM = ot.FORM(OptAlgo, eventF,
startingPoint)
algoFORM.run()

#SORM
OptAlgo = ot.Cobyla()
startingPoint = Distribution.getMean()
algoSORM = ot.SORM(OptAlgo, eventF,
startingPoint)
algoSORM.run()
```

Different types and parameterizations of finite difference gradient computation are available

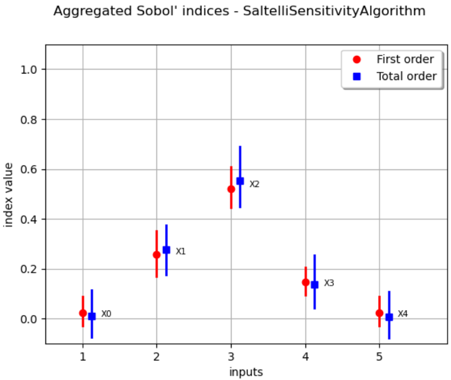
Also:

- Directional sampling
- Importance sampling (FORM-IS, NAIS, Adaptive IS-Cross-entropy)
- Subset sampling

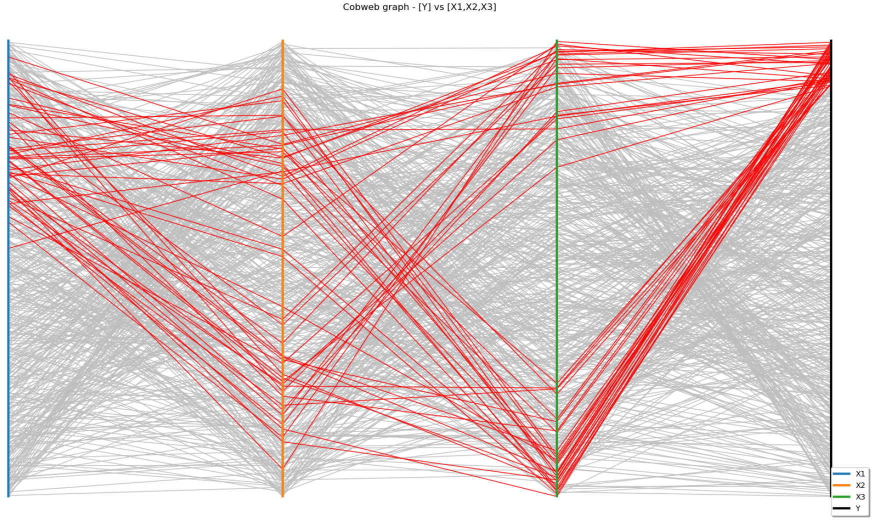
Sensitivity analysis

Various sensitivity analysis methods are available

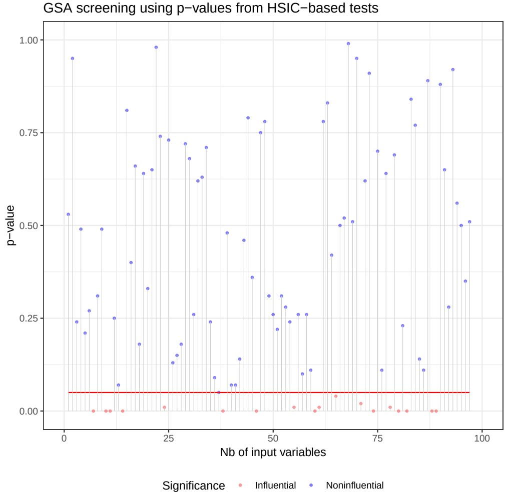
- Graphical analysis
  - Pair plots
  - Parallel coordinates plots
  - Cross-cuts
- Quantitative indices
  - SRC, SRRC, PRC, PRCC importance measures
  - Sobol' indices (several estimators)
  - FAST indices
  - ANCOVA indices
  - HSIC indices
  - Shapley effects (available as a module)



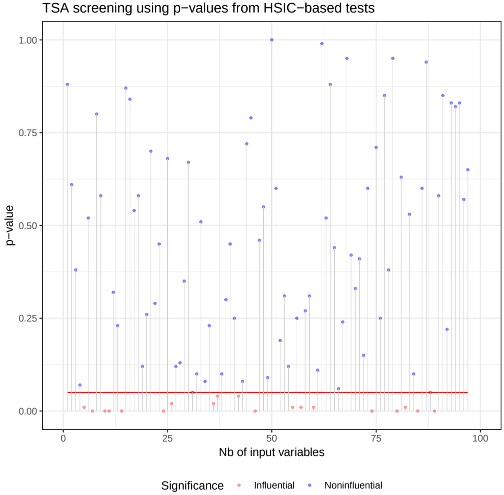
Sensitivity analysis: Parallel coordinates plot



Sensitivity analysis: HSIC indices and associated p-values



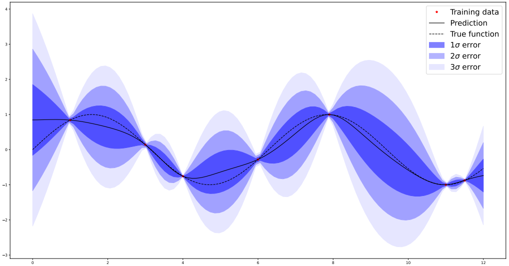
GSA-oriented screening.



TSA-oriented screening.

Surrogate modeling: Gaussian process regression (Kriging)

- Different surrogate modeling methods are available
  - Kriging
  - Polynomial chaos expansion
  - Linear regression & step-wise basis selection
  - Low rank tensors
  - automatic validation tools



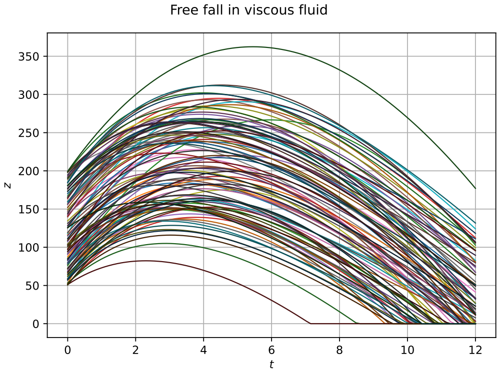
- Gaussian process regression
  - Different types of covariance functions and function basis can be used
  - User-defined options are also available
  - MLE optimization can be parameterized
  - Large number of optimization algorithms available

```
inputSample = Distribution.getSample(100)
outputSample = fun(inputSample)

dimension = 4
basis = ot.ConstantBasisFactory(dimension).build()
covarianceModel = ot.SquaredExponential()

algo = ot.KrigingAlgorithm(inputSample, outputSample,
     covarianceModel, basis)

algo.run()
result = algo.getResult()
KrigingMM = result.getMetaModel()
```

Optimization

- OpenTURNS provides an interface with several optimization libraries
  - Bonmin
  - NLopt
  - dlib
  - pagmo
- Ad-hoc implementation of the COBYLA algorithm
- Constrained and unconstrained optimization
- Gradient-based and derivative-free optimizaiton
- Bound and unbound optimization
- Single and multi-objective optimization
- Multi-start wrapper

Field function modeling

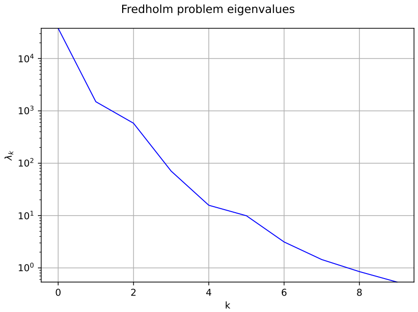



```python
def FreeFall(X):
    g  = 9.81
    z0,v0,m,c = X
    tau=m/c
    vinf=-m*g/c
    t = np.array(mesh.getVertices().asPoint())
    z=z0+vinf*t+tau*(v0-vinf)*(1-np.exp(-t/tau))
    z=np.maximum(z,0.0)
    return ot.Field(mesh, [[zeta] for zeta in z])

tmin=0.
tmax=12.
gridsize=100
mesh = ot.IntervalMesher([gridsize-1]).build(
ot.Interval(tmin, tmax))

alti = ot.PythonPointToFieldFunction(4, mesh,  1, AltiFunc)

distZ0 = ot.Uniform(50.0, 200.0)
distV0 = ot.Normal(55.0, 10.0)
distM = ot.Normal(80.0, 8.0)
distC = ot.Uniform(0.0, 30.0)
distX = ot.ComposedDistribution([distZ0, distV0,
 distM, distC])

size = 100
inputSample = distX.getSample(size)
outputField = alti(inputSample)
```

Dimension reduction: Karhunen-Loeve decomposition

- We wish to reduce the dimension of the problem from a infinite dimensional output to a finite dimensional one
- We can perform a Karhunen-Loeve decomposition with a finite truncature
- This requires to solve a Fredholm's problem in order to identify the eigenfunctions and associated eigenvalues of the considered process

$$Y(\omega,\underline{t}) = \sum_{k=1}^{\infty} \sqrt{\lambda_k}\xi_k(\omega)\underline{\varphi}_k(\underline{t}) \rightarrow \tilde{Y}(\omega,\underline{t}) = \sum_{k=1}^{p} \sqrt{\lambda_k}\xi_k(\omega)\underline{\varphi}_k(\underline{t})$$

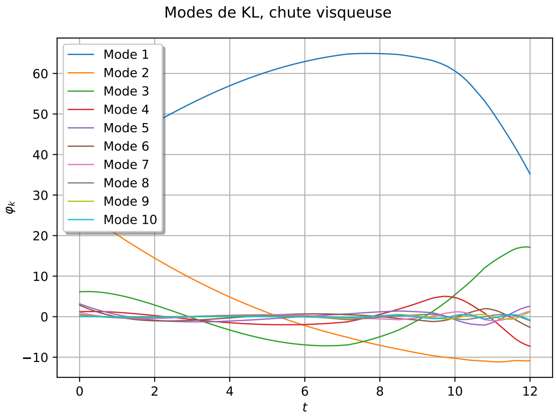



```
meanFunction = ot.P1LagrangeEvaluation(
 meanField)
trend = ot.TrendTransform(meanFunction, myMesh)
invTrend = trend.getInverse()
outputFieldCentered = invTrend(outputField)

truncThreshold = 1.0e-5
algo = ot.KarhunenLoeveSVDAlgorithm(
 outputFieldCentered, truncThreshold)
algo.run()
KLResult = algo.getResult()

eigenValues = KLResult.getEigenValues()
```

Dimension reduction: Karhunen-Loeve decomposition

$$\tilde{Y}(\omega, \underline{t}) = \sum_{k=1}^{p} \sqrt{\lambda_k} \xi_k(\omega) \underline{\varphi}_k(\underline{t})$$

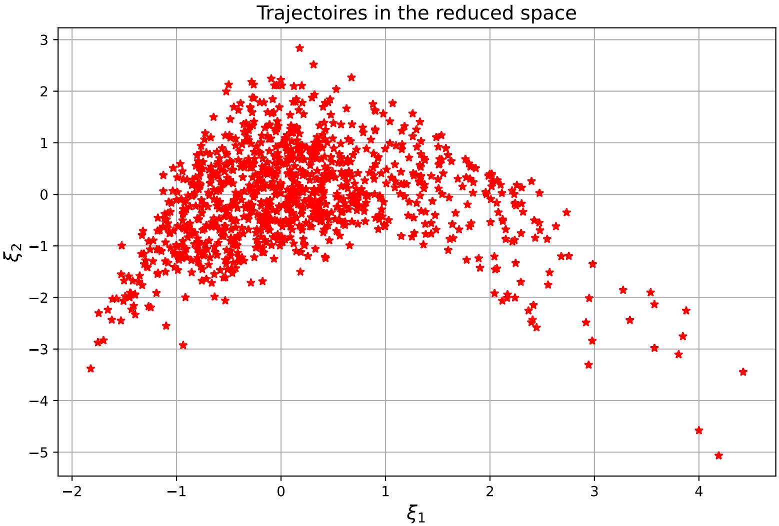
Main modes:




```
scaledModes =
 KLResult.getScaledModesAsProcessSample()
graph = scaledModes.drawMarginal(0)
graph.setTitle('Modes de KL, chute visqueuse')
graph.setXTitle(r'$t$')
graph.setYTitle(r'$\varphi_k$')
leg = ot.Description([ 'Mode '+str(i +1) for
  i in range(eigenValues.getDimension()) ])
graph.setLegends(leg)
graph.setLegendPosition('topleft')
view=View(graph)
```

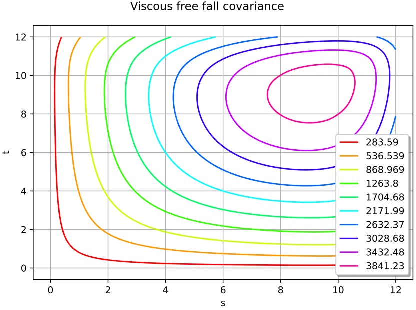
Dimension reduction: Karhunen-Loeve decomposition

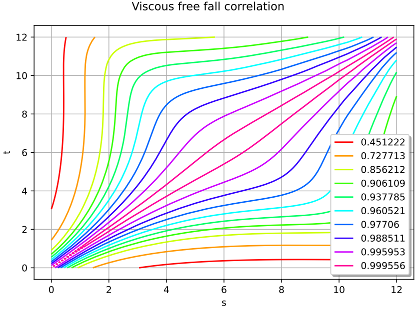
We only consider the first 2 terms of the decomposition:



Field function analysis

We center the trajectories with respect to the mean field:

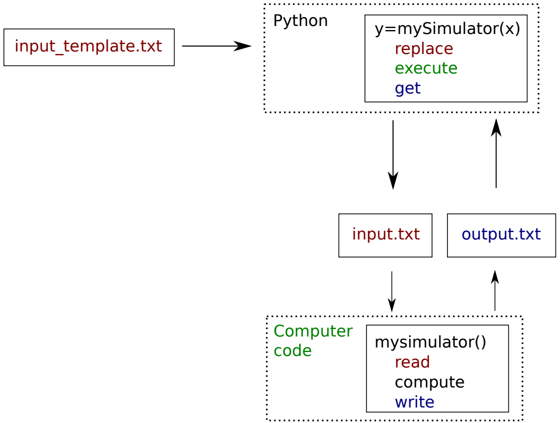





```
cov = KLResult.getCovarianceModel()

# As a covariance function
isStationary = False
asCorrelation = False
graph = cov.draw(0, 0, tmin, tmax, 128, isStationary, asCorrelation)

# As a correlation function
asCorrelation = True
graph = cov.draw(0, 0, tmin, tmax, 128, isStationary, asCorrelation)
```

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

Support, discussion and contribution

- GitHub repository: `https://github.com/openturns/openturns`
  - Bug report
  - Enhancement suggestions
  - Contribute
  - Review contributions
- Discourse forum: `https://openturns.discourse.group/`
  - Practical questions
  - Theoretical questions
  - Feature request
  - Forum layout
- Gitter chat: `https://gitter.im/openturns`
  - Practical questions
  - Theoretical questions
  - Feature request
  - Chat layout

A few recent highlights

- Introduction of the **experimental** sub-module
- New services
  - Introduction of generalized extreme value distributions
  - Cross-entropy importance sampling & Non-parametric adaptive importance sampling
  - Uniform sampling on a mesh
  - Field to vector surrogate modeling & sensitivity
  - Iterative statistics (mean, variance, Sobol' indices)
  - HSIC indices
  - New examples and use-cases
  - ...
- Performance enhancement
  - Parallelization and optimized computation of HSIC indices & p-values
  - Improved interface and flexibility of the Metropolis-Hastings sampling classes
  - Improved polynomial chaos expansion API
  - Coupling with the Pagmo optimization library
  - ...

Reliability analysis: Cross-Entropy Importance Sampling

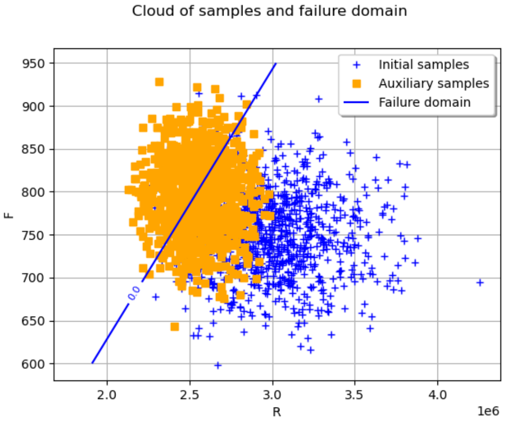
We wish to evaluate the probability of a given event through Importance Sampling:

$$\hat{P}_{\text{IS}} = \frac{1}{N} \sum_{i=1}^{N} 1_{g(\mathbf{x}_i)<T} \frac{f_{\mathbf{X}}(\mathbf{x}_i)}{h(\mathbf{x}_i)}$$

- $f_{\mathbf{X}}$ input distribution
- h parametric auxiliary distribution
- $\mathbf{x}_i$ generated according to h
- h is updated during the sampling process so as to tend towards its optimal value
- We can work in both the **physical** and the **standard** Gaussian spaces

Reliability analysis: Cross-Entropy Importance Sampling

Sampling in the standard Gaussian space

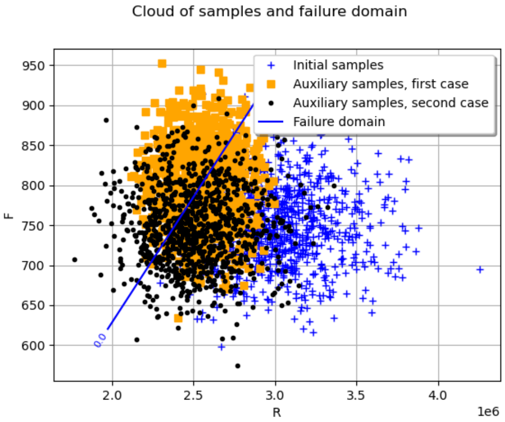



```
Y = ot.CompositeRandomVector(g, X)
event = ot.ThresholdEvent(Y, ot.Less(), 0.0)

# We choose to set the intermediate quantile level to
    0.35.

standardSpaceIS = otexp.
    StandardSpaceCrossEntropyImportanceSampling(event,
    0.35)

# The sample size at each iteration can be changed
standardSpaceIS.setMaximumOuterSampling(1000)

standardSpaceIS.run()
standardSpaceISResult = standardSpaceIS.getResult()
```

- Probability of failure: 0.029465848610494363
- Coefficient of variation: 0.045029988245260714

Reliability analysis: Cross-Entropy Importance Sampling

Sampling in the physical space



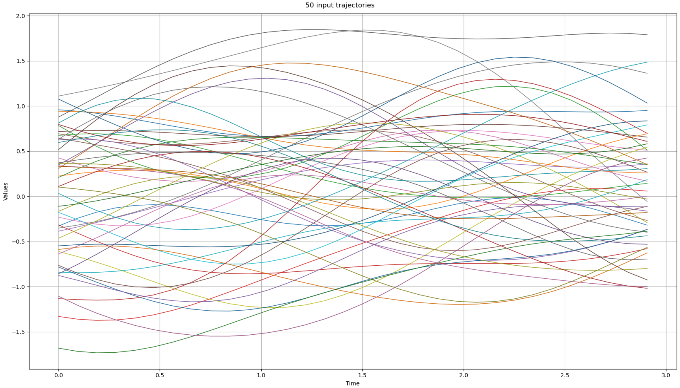

```
marginR = ot.LogNormalMuSigma().getDistribution()
marginF = ot.Normal()
auxiliaryDistribution = ot.ComposedDistribution([marginR
      , marginF])

# Case 1: optimize al parameters
physicalSpaceIS1 = otexp.
     PhysicalSpaceCrossEntropyImportanceSampling(
    event, auxiliaryDistribution, activeParameters,
     initialParameters, bounds
)
physicalSpaceIS1.run()

# Case 2: only distribution means are optimized
activeParameters = ot.Indices([0, 3])
physicalSpaceIS2 = otexp.
     PhysicalSpaceCrossEntropyImportanceSampling(
    event, auxiliaryDistribution, activeParameters,
     initialParameters, bounds
)
physicalSpaceIS2.run()
```

- Probability of failure: 0.029702353119720654
- Coefficient of variation: 0.04321365594527282

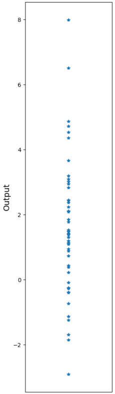
Applications with functional inputs

- We consider here an application which takes a random field as input, and provides a vectorial output:

$$h : \left| \begin{array}{ccc} \mathcal{M}_N \times (\mathbb{R}^d)^N & \rightarrow & \mathbb{R}^p \\ \mathbf{X} & \rightarrow & \mathbf{Y} \end{array} \right.$$

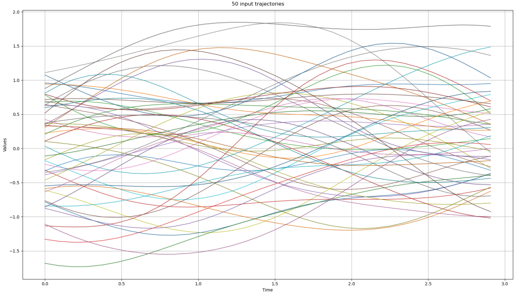


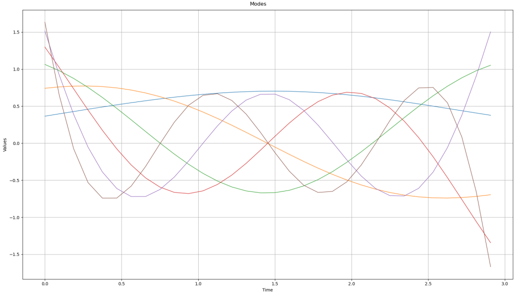
$\rightarrow$



Applications with functional inputs

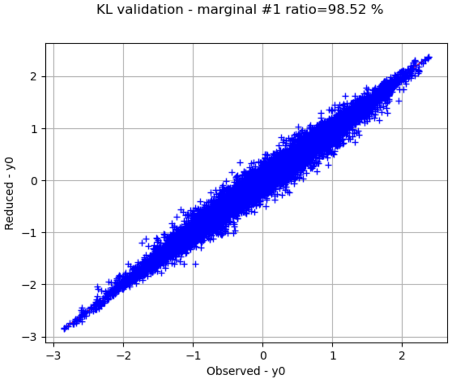
- We want to create a surrogate model, $\tilde{h}$, of the function at hand
- The class **FieldToPointFunctionalChaosAlgorithm** allows to do so by combining the following functionalities:
  - Karhunen-Loeve decomposition of the functional input over a discrete mesh
  - Creation of a polynomial chaos expansion surrogate model between the resulting reduced space and the vectorial outputs





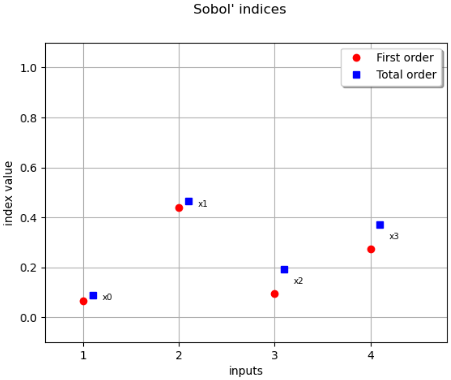
Applications with functional inputs

```
algo = otexp.FieldToPointFunctionalChaosAlgorithm(x, y)
# 1. KL parameters
algo.setThreshold(4e-2)  # we expect to explain 96% of
     variance
algo.setNbModes(10)  # max KL modes (default=unlimited)
algo.run()
result = algo.getResult()
kl_results = result.getInputKLResultCollection()

for i in range(x.getDimension()):
    validation = ot.KarhunenLoeveValidation(x.
     getMarginal(i), kl_results[i])
    graph = validation.drawValidation().getGraph(0, 0)
```



Applications with functional inputs

```
sensitivity = otexp.FieldFunctionalChaosSobolIndices(
     result)
graph = sensitivity.draw()
```

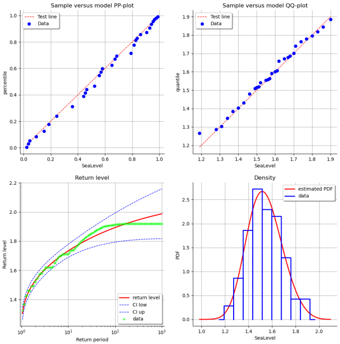


Modeling generalized extreme value (GEV) distributions

Estimation techniques

- Likelihood maximization
- Profile likelihood maximization

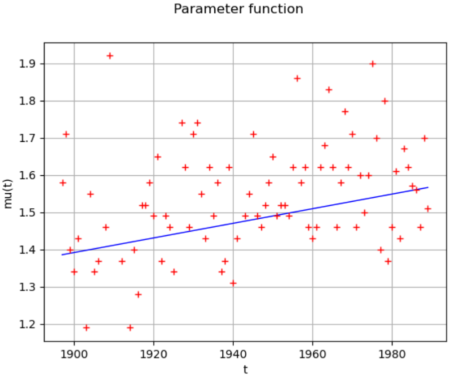
Estimation of a return level (sea level data)



Stationary and non-stationary data

- The GEV parameters can be made to vary as a function of time:

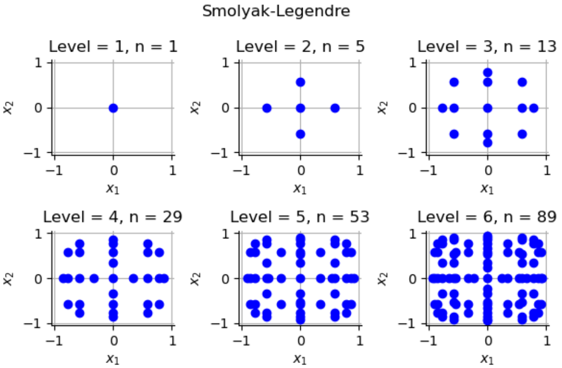
$$\mathrm{GEV}(\mu(t), \sigma(t), \xi(t))$$



Efficient numerical quadrature: Smolyak-Legendre quadrature

- Combines multi-dimensional Gauss-Legendre quadrature with an efficient selection of the polynomial multi-indices.

```
uniform = ot.GaussProductExperiment(ot.Uniform(-1.0,
     1.0))
collection = [uniform] * 2

number_of_rows = 2
number_of_columns = 3
bounding_box = ot.Interval([-1.05] * 2, [1.05] * 2)
grid = ot.GridLayout(number_of_rows, number_of_columns)
for i in range(number_of_rows):
    for j in range(number_of_columns):
        level = 1 + j + i * number_of_columns
        experiment = otexp.SmolyakExperiment(collection,
      level)
        nodes, weights = experiment.generateWithWeights
     ()
        sample_size = weights.getDimension()
```

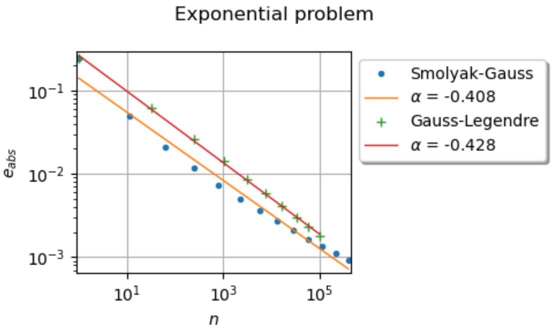


Efficient numerical quadrature: Smolyak-Legendre quadrature

Example on the integration of:

$$g(\mathbf{x}) = (1 + 1/d)^d \prod_{i=1}^{d} x_i^{1/d}$$

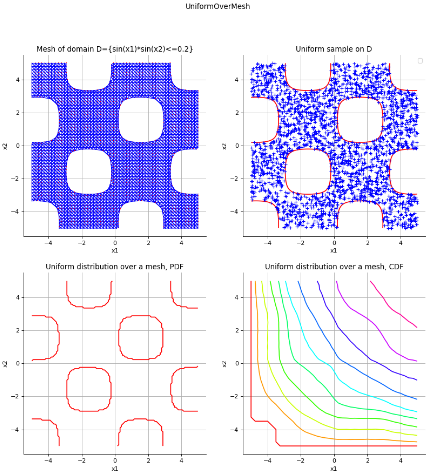
```
uniform = ot.GaussProductExperiment(ot.Uniform(0.0, 1.0)
     )
collection = [uniform] * dimension
level = 5
print("level = ", level)
experiment = otexp.SmolyakExperiment(collection, level)
nodes, weights = experiment.generateWithWeights()

g_values = g_function(nodes)
g_values_point = g_values.asPoint()
approximate_integral = g_values_point.dot(weights)
lre10 = -np.log10(abs(approximate_integral - integral) /
      abs(integral))
```



Sampling on a mesh

We wish to sample over a mesh, proportionally to the size of each cell, and uniformly in a given cell

```
f = ot.SymbolicFunction(['x', 'y'], ['sin(x)*sin(y)'])
levelSet = ot.LevelSet(f, ot.Less(), 0.2)
box = ot.Interval([-5.0]*2, [5.0]*2)
mesh = ot.LevelSetMesher([50]*2).build(levelSet, box,
     False)
distribution = otexp.UniformOverMesh(mesh)

sample = distribution.getSample(5)

mesh = distribution.getMesh()
algo = distribution.getIntegrationAlgorithm()
distribution.setIntegrationAlgorithm(ot.GaussLegendre
     ([10] * 2))
```



PERSALYS, the graphical user interface of OpenTURNS

- Provides 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, linear regression), screening (Morris), optimization, design of experiments
- GUI language: English, French
- Partners: EDF, Phiméca
- Licence: LGPL
- OS: Windows and Linux
- Schedule: Since summer 2016, two releases per year, currently V13

https://www.persalys.fr/

Persalys

The end

Thanks!

Any question?