Overview of OpenTURNS and its graphical user interface Persalys

J. Muré [1] M. Baudin [1] J. Pelamatti [1]
J. Schueller [2] A. Marion [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, schueller@phimeca.com

June 17th 2025, BEPU 2025, Rhodes Island, Greece

Contents

OpenTURNS: www.openturns.org

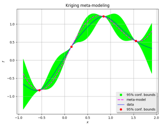
OpenTURNS

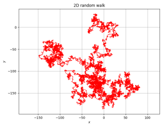
An Open source initiative for the Treatment of Uncertainties, Risks'N Statistics

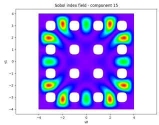
- Generic coupling to any type of physical model
- Open source, LGPL licensed, C++/Python library
- Linux, Windows, macOS. First release : 2007
- on average 4 full time developers

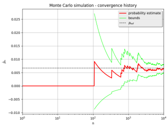
OpenTURNS: content

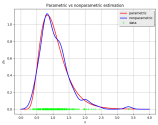
- Data analysis
  - Distribution fitting
  - Statistical tests
  - Estimate dependency and copulas
  - Estimate stochastic processes
- Probabilistic modeling
  - Dependence modeling
  - Univariate distributions
  - Multivariate distrbutions
  - Copulas
  - **Conditional distributions**
  - Processes
  - 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







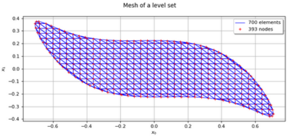




OpenTURNS: documentation

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 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 level set type.
`getClassName`()	Accessor to the object's name.
`getDiscretization`()	Accessor to the discretization.

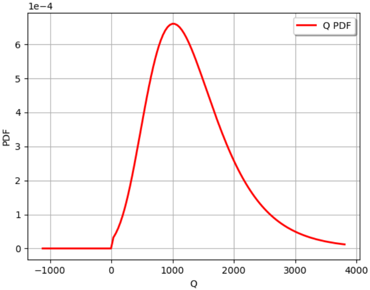
- Content:
  - Programming interface (API)
  - Examples
  - Theory
- *All* classes and methods are documented, partly automatically.
- Examples are automatically tested at *each* update of the code and outputs are checked.

OpenTURNS features

- C++ and Python interface
- Parallel computations with shared memory
- Optimized linear algebra with LAPACK and BLAS
- Possibility to interface with a SLURM-based HPC cluster now made easier with the helper library *othpc* available at `https://github.com/openturns/othpc` (soon to be published as a pip/conda package)
- Installation through conda, pip, packages for various Linux distributions and source code
-
```python
import numpy as np
import openturns as ot
from openturns.viewer import View
ot.ResourceMap.SetAsString("Contour-DefaultColorMap", "viridis")
ot.ResourceMap.SetAsBool("Contour-DefaultIsFilled", True)
ot.ResourceMap.SetAsUnsignedInteger("Contour-DefaultLevelsNumber", 15)
```

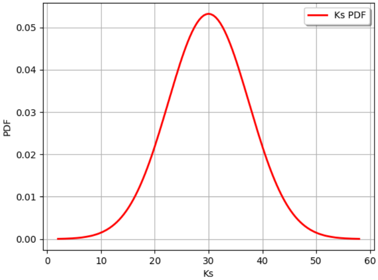
Probabilistic modeling

Gumbel(scale=558, mode=1013) truncated to $[0, +\infty)$




```
d = ot.Gumbel(558.0, 1013.0)
tr = ot.TruncatedDistribution.LOWER
Q = ot.TruncatedDistribution(d, 0.0, tr)
```

Normal(mean=30, std=7.5) truncated to $[0, +\infty)$



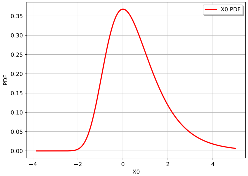

```
d = ot.Normal(30.,7.5)
tr = ot.TruncatedDistribution.LOWER
Ks = ot.TruncatedDistribution(d, 0.0, tr)
```

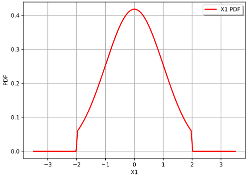
Probabilistic modeling

We consider a 2-dimensional distribution with the following marginals:

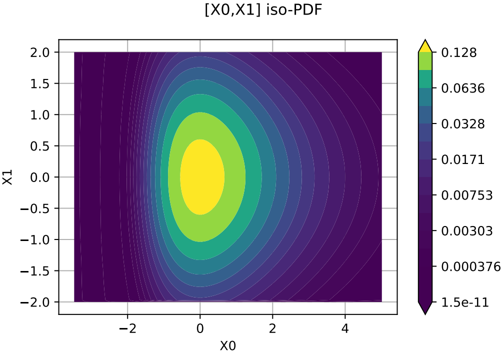
- Gumbel(scale = 1, mode = 0)
- Truncated normal (mean = 0, std = 1, min = -2, max = 2)

Marginals

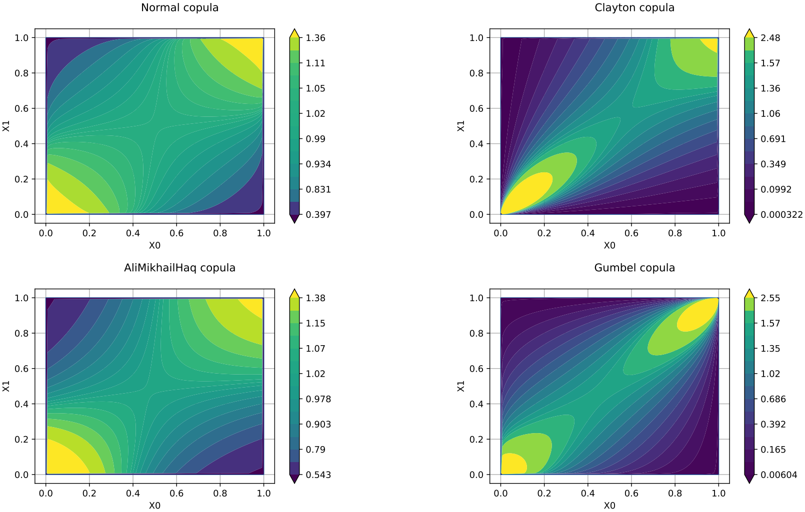




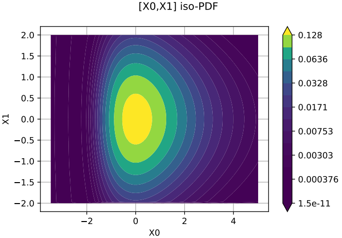
Joint distribution

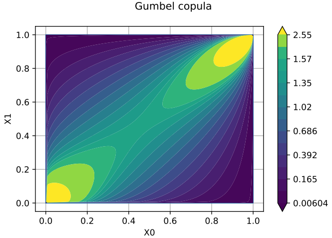


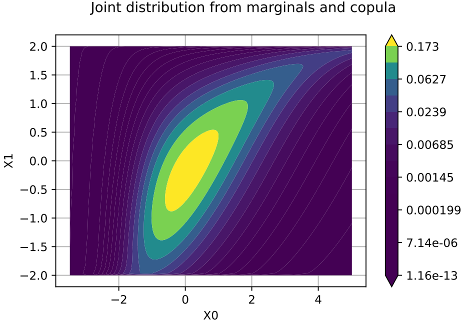
Beyond independent marginals: Copulas



Composing marginal distributions and copulas



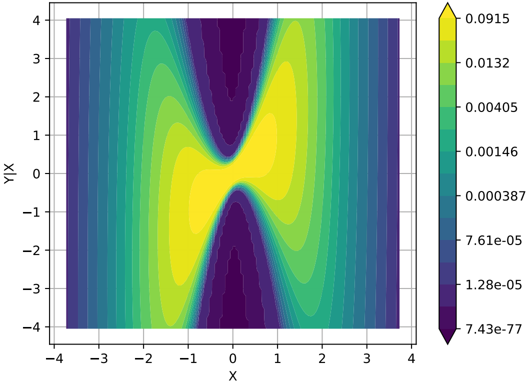





```python
marg1 = ot.Gumbel(1.0, 0.0)
marg2 = ot.TruncatedNormal(0.0, 1.0, -2.0, 2.0)
copula = ot.GumbelCopula()
joint = ot.JointDistribution([marg1, marg2],
     copula)
graph = joint.drawPDF()
title = "Joint distribution "
title += "from marginals and copula"
graph.setTitle(title)
```

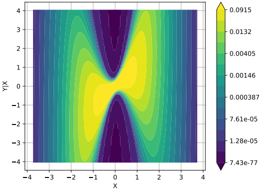
New feature: Joint distribution defined from conditionals

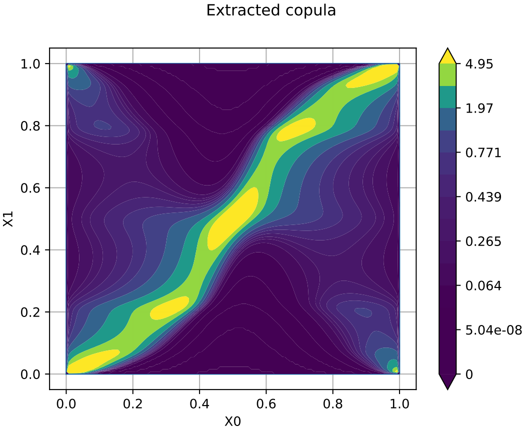
$$Y|X \sim \mathcal{N}(X, 0.1 + X^2)$$
$$X \sim \mathcal{N}(0, 1)$$




```
Xdist = ot.Normal(0.0, 1.0)
f = ot.SymbolicFunction(["x"],["x", "0.1 + x^2"])
Ycond = ot.Normal()
XYdist = ot.JointByConditioningDistribution(Ycond, Xdist, f)
ot.ResourceMap.SetAsString("Contour-DefaultColorMapNorm", "rank")
graph = XYdist.drawPDF()
graph.setTitle("")
graph.setXTitle("X")
graph.setYTitle("Y|X")
```

Extract the copula

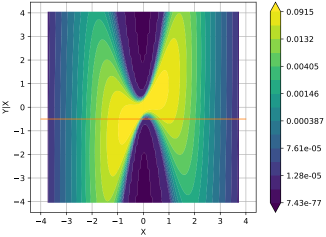




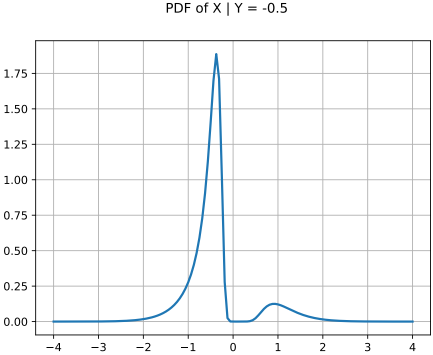

```
copula = XYdist.getCopula()
ot.ResourceMap.SetAsString("Contour-DefaultColorMapNorm", "linear")
graph = copula.drawPDF()
graph.setTitle("Extracted copula")
```

New feature: Conditional from joint distribution

$$Y|X \sim \mathcal{N}(X, 0.1 + X^2)$$
$$X \sim \mathcal{N}(0, 1)$$



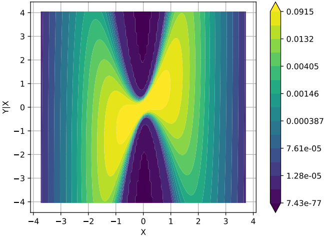
$$X|Y \sim ?$$



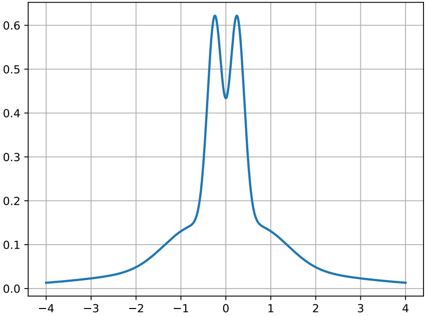

```
from openturns.experimental import PointConditionalDistribution
cond_dist = PointConditionalDistribution(XYdist, [1], [-0.5]) # Y is set to -0.5
graph = cond_dist.drawPDF(-4.0, 4.0)
graph.setXTitle(""); graph.setYTitle(""); graph.setLegends([""])
graph.setTitle("PDF of X | Y = -0.5")
```

New feature: Compound distribution

$$Y|X \sim \mathcal{N}(X, 0.1 + X^2)$$
$$X \sim \mathcal{N}(0, 1)$$



$$Y \sim ?$$



```
Ydist = ot.DeconditionedDistribution(Ycond, Xdist, f) # Compound distribution
graph = Ydist.drawPDF(-4.0, 4.0, 1000) # draw from -4 to 4 with 1000 points
graph.setLegends([""]); graph.setXTitle(""); graph.setYTitle(""); graph.setTitle("")
```

New feature: Posterior distribution

$$Y_1, \ldots, Y_n | X \sim_{iid} \mathcal{N}(X, 0.1 + X^2) \qquad \text{(likelihood)}$$

$$X \sim \mathcal{N}(0, 1) \qquad \text{(prior)}$$

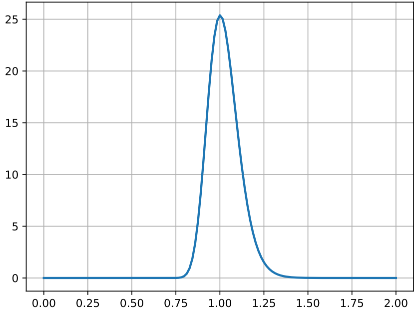
$$X | Y_1, \ldots, Y_n \sim ? \qquad \text{(posterior)}$$

Setting $X = 1$, sample $Y_1, \ldots, Y_{20}$ according to (likelihood).

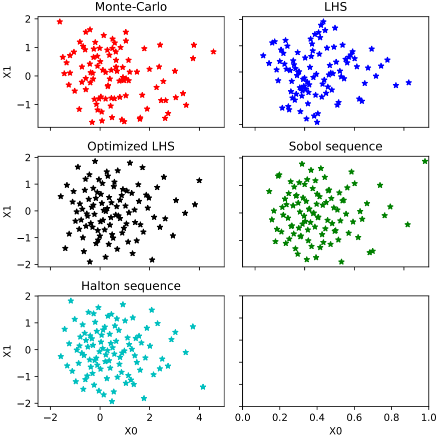
```
X = 1.0
# for reproducibility
ot.RandomGenerator.SetSeed(0)

Ysam=ot.Normal(X,0.1+X**2).getSample(20)
```

Draw the PDF of (posterior).

```
from openturns.experimental import
     PosteriorDistribution
post = PosteriorDistribution(Ydist,Ysam)
graph = post.drawPDF(0.0, 2.0)
graph.setXTitle("");graph.setYTitle("");
graph.setTitle("")
graph.setLegends([""])
```



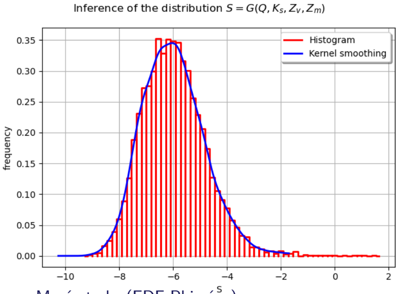
Design of experiments




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

sample1 = dist.getSample(sampleSize)

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

lhs = ot.LHSExperiment(dist, 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) # or Halton
experiment = ot.LowDiscrepancyExperiment(
  sequence, dist, 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.JointDistribution([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)
hist = ot.HistogramFactory().build(sampleY)
graph = hist.drawPDF()
distKS = ot.KernelSmoothing().build(sampleY)
graph2 = distKS.drawPDF()
```

New feature: Quantile confidence intervals

Let $(X^{(1)} \ldots, X^{(N)})$ be order statistics of a random variable X: $X^{(1)} \leq \cdots \leq X^{(N)}$.

Given the quantile level $\alpha \in [0, 1]$ and the confidence level $\beta \in [0, 1]$, we seek ranks $1 \leq k \leq k' \leq N$ such that the quantile x_α of X verifies:

$$\mathbb{P}\left(X^{(k)} \leq x_\alpha \leq X^{(k')}\right) \geq \beta.$$

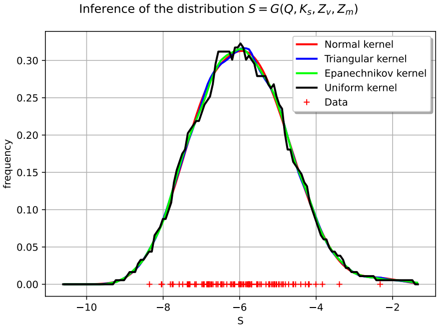
Bilateral quantile confidence interval:

```
from openturns.experimental import QuantileConfidence
n = 400; alpha = 0.05; beta = 0.95
q = QuantileConfidence(alpha, beta)
i_n, j_n = q.computeBilateralRank(n)
ot.RandomGenerator.SetSeed(0)
sam = ot.Gumbel().getSample(n)
bci = q.computeBilateralConfidenceInterval(sam)
print(f"ranks={[i_n, j_n]} CI={bci}")

Out: ranks=[11, 28] CI=[-1.36431, -0.981749]
```

Unilateral quantile confidence interval:

```
k_n = q.computeUnilateralRank(n, True) # True means we want a finite lower bound
uci = q.computeUnilateralConfidenceInterval(sam, True)
print(f"rank={k_n} CI={uci}")

Out: rank=12 CI=[-1.35859, (1.79769e+308) +inf[
```

Distribution and dependence inference



- Parametric $(1d - Nd)$ distribution inference
- Non-parametric $(1d - Nd)$ distribution inference
- Parametric copula inference
- Non-parametric copula inference (Bernstein copula)
- Resampling from inferred distributions

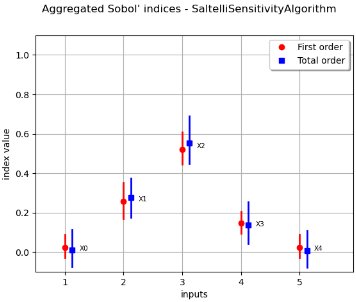
```
samY = outputVector.getSample(100)
graph=ot.KernelSmoothing(ot.Normal()).build(samY).drawPDF()
distKS = ot.KernelSmoothing(ot.Triangular()).build(samY)
graph2 = distKS.drawPDF()
graph.add(graph2)
distKS = ot.KernelSmoothing(ot.Epanechnikov()).build(samY)
graph2 = distKS.drawPDF()
graph.add(graph2)
distKS = ot.KernelSmoothing(ot.Uniform()).build(samY)
graph2 = distKS.drawPDF()
```

Sensitivity analysis

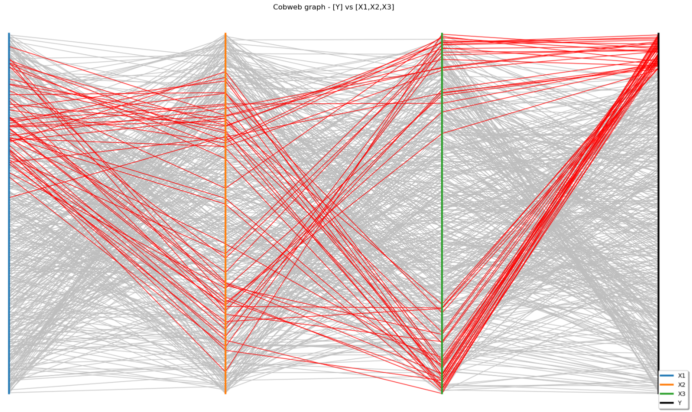
Various sensitivity analysis methods are available

- Graphical analysis
  - Pair plots
  - Parallel coordinates plots
  - Cross-cuts

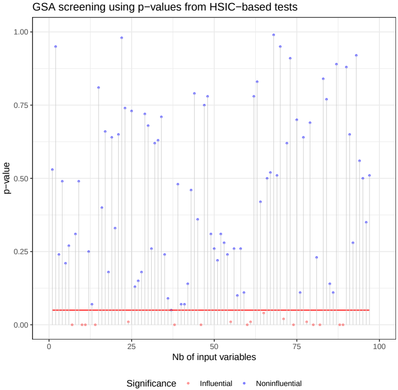
- Quantitative indices
  - SRC, SRRC, PRC, PRCC
  - Sobol' indices (multiple estimators)
  - FAST indices
  - ANCOVA indices
  - HSIC indices
  - Shapley Indices (available as a separate library: *otshapley*)



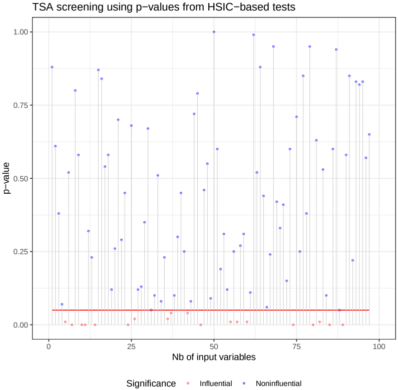
Sensitivity analysis: Parallel coordinates plot



Sensitivity analysis: HSIC indices and associated p-values



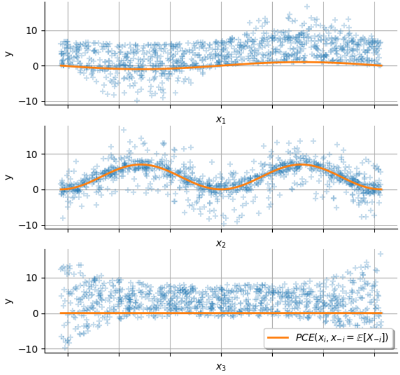
GSA-oriented screening.



TSA-oriented screening.

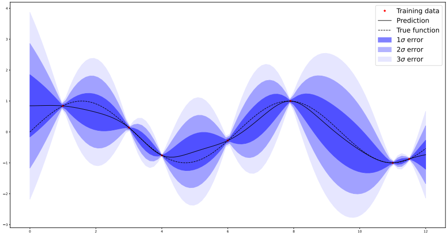
Surrogate models

- Polynomial chaos expansion



- PCE highlight: extract the conditional expectation with respect to any variables

- Gaussian process regression

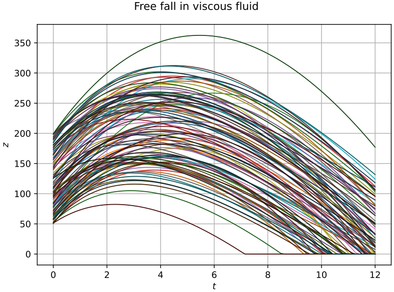



```python
import openturns.experimental as exp
inputSample = Distribution.getSample(100)
outputSample = fun(inputSample)
basis = ot.ConstantBasisFactory(1).build()
covarianceModel = ot.MaternModel()

fit = exp.GaussianProcessFitter(inputSample,
      outputSample, covarianceModel, basis)
fit.run()
fitted = fit.getResult()
algo = exp.GaussianProcessRegression(fitted)
algo.run()
result = algo.getResult()
surrogate = result.getMetaModel()
```

Optimization

- OpenTURNS provides an interface with several optimization libraries
  - Bonmin
  - NLopt
  - dlib
  - pagmo
- Constrained and unconstrained optimization
- Gradient-based and derivative-free optimizaiton
- Bounded and unbounded optimization
- Single and multi-objective optimization
- Multi-start wrapper

Field function modeling

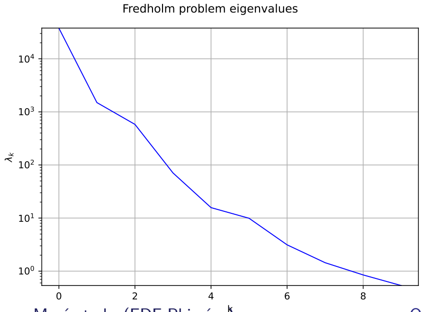



```
def free_fall(X):
    g  = 9.81
    z0,v0,m,c = X
    tau=m/c
    vinf=-m*g/c
    t = np.array(mesh.getVertices().asPoint())
    z=z0+vinf*t+tau*(v0-vinf)*(1-np.exp(-t/tau))
    z=np.maximum(z,0.0)
    return ot.Field(mesh, z.reshape(-1, 1))

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

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

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.JointDistribution([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 finite truncature.
- This requires to solve a Fredholm's problem in order to identify the eigenfunctions and associated eigenvalues of the considered process.

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

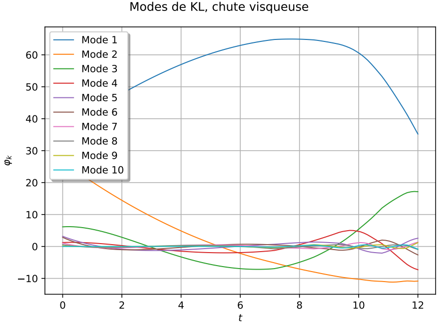



```
meanField = outputField.computeMean()
meanFunc = ot.P1LagrangeEvaluation(meanField)
trend = ot.TrendTransform(meanFunc, mesh)
invTrend = trend.getInverse()
outputFieldCentered = invTrend(outputField)

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

eigenValues = KLResult.getEigenvalues()
```

Dimension reduction: Karhunen-Loeve decomposition

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

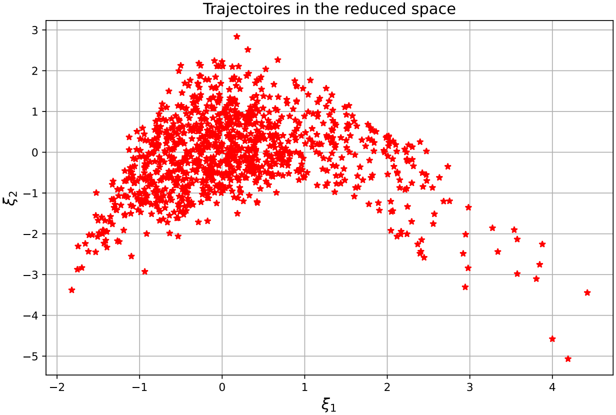
Main modes:




```
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

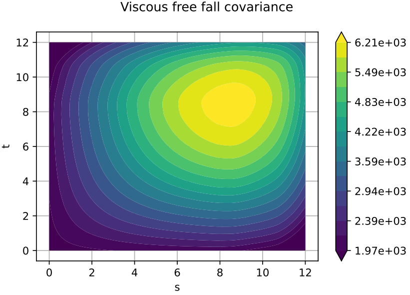
We only consider the first 2 terms of the decomposition:

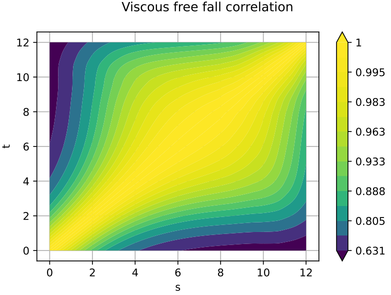



```
projectionFunction = ot.KarhunenLoeveProjection(KLResult)
sampleKsi = projectionFunction(outputFieldCentered)
sampleKsi = sampleKsi[:,:2]
```

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)
```

Contents

Project overview

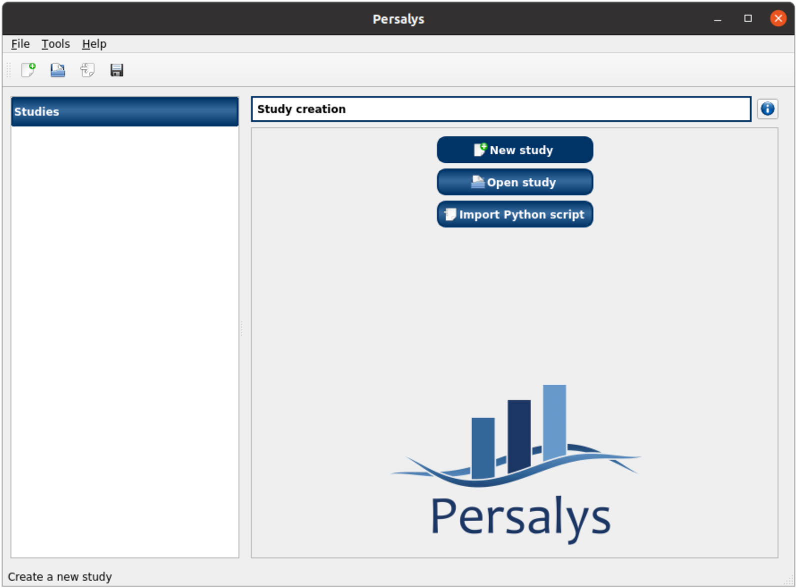
- Partnership between EDF and Phimeca since 2015
  - Developed in C++ using Qt
  - Aiming at maximizing the use of OpenTURNS - through a dedicated GUI - for engineers/researchers without a strong coding experience
  - As easy to use as possible while providing the user with help and guidelines
  - Benefit from the advanced visualization capability of Paraview

- Features:
  - Uncertainty quantification:
  - Probabilistic model definition
  - Distribution fitting
  - Probability estimate
  - Metamodeling
  - Screening
  - Optimization
  - Design of experiments
  - As generic as possible
  - Allows for a wide variety of models
  - Can be coupled to external code
  - GUI language in both English and French
- LGPL license
- Two releases per year, follows OpenTURNS development
- Available for free on demand at `https://persalys.fr`

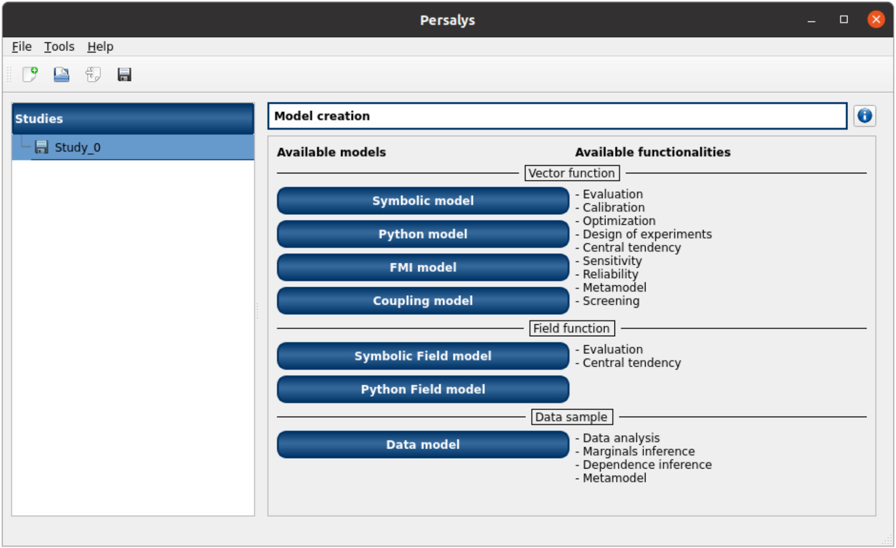
Persalys installation

- Github: sources
- You can request executables at `https://persalys.fr/obtenir.php?la=en`
- Depending on your OS
  - Linux → .AppImage (600 Mo)
  - Windows → .exe will create a shortcut on your Desktop (program is 1.45 Go)
- Also distributed by Debian-based GNU/Linux distributions (e.g. Debian, Ubuntu...)

Open Persalys and click on “New study”



Create a study



Definition/Evaluation

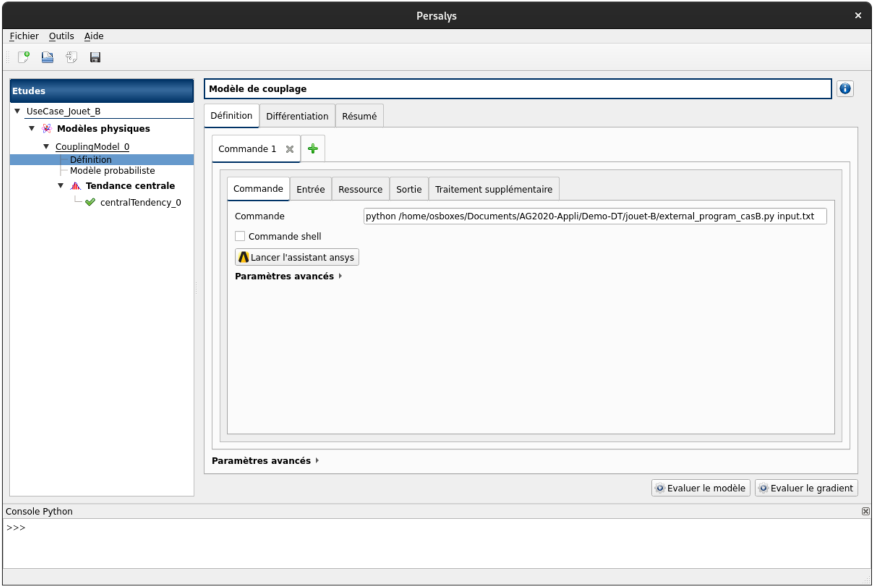
Models are viewed as “black boxes” with specific inputs and outputs. Persalys supports two model categories:

- vector to vector ($X_i \to Y_i$, emphasized here)
- vector to 1D field ($X_i \to Y_i(t)$)

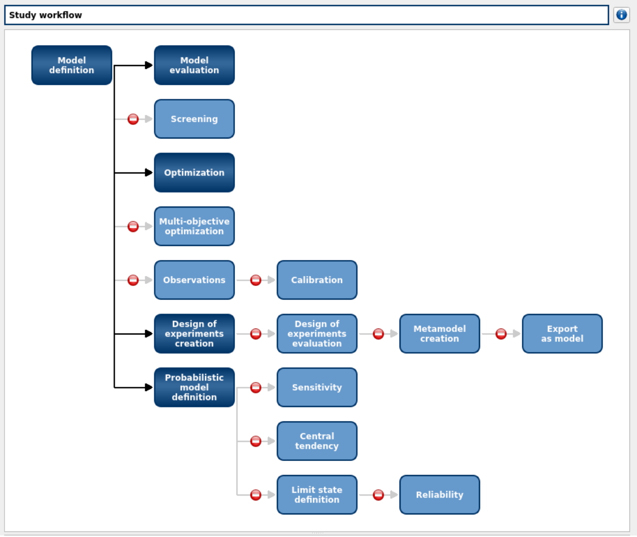
Vector to vector models can be of the following type:

- Symbolic model (i.e. a mathematical formula)
- Python model (i.e. a Python function)
- FMI model
- Coupling model (an executable command which reads/writes input/output must be provided) files

Coupling model definition



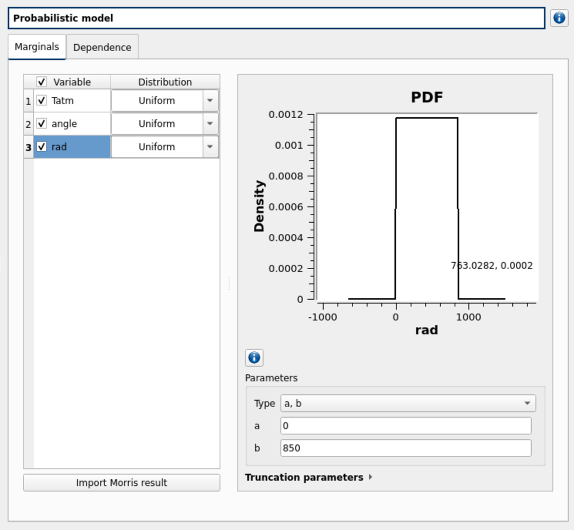
Study workflow



Blocks become available as study content grows and prerequisites are fulfilled.

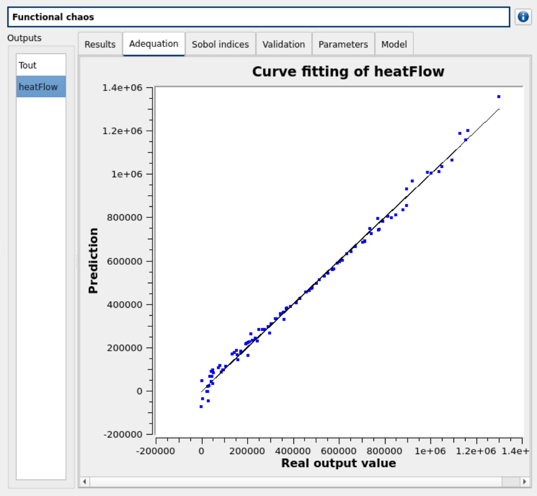
Probabilistic models

Each input variable can be associated to a distribution.
Dependencies between variables are specified as copulas.



Surrogate model creation

Using an evaluated design of experiments, the user can build a surrogate model (linear regression, functional chaos or Gaussian process regression). Validation tests are run to check the approximation.



OpenTURNS resources

- Website and documentation: `www.openturns.org`
- GitHub: `https://github.com/openturns/openturns`
- Forum: `https://openturns.discourse.group`

Persalys resources

- Website and documentation: `https://persalys.fr/?la=en`
- GitHub: `https://github.com/persalys/persalys`
- Forum: `https://persalys.discourse.group`