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

May 22nd 2024, BEPU 2024, Lucca, Italy

Contents

OpenTURNS Overview

Persalys Overview

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, macOS
- ▶ First release : 2007
- ▶ 5 full time developers
- ▶ Users $\approx$ 1000, mainly in France (1 078 000 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

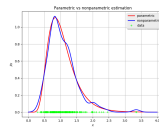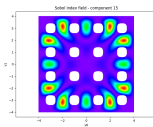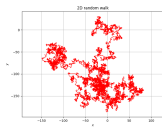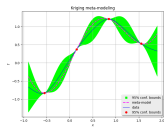- **Reliability, sensitivity**
  - Sampling methods
  - Approximation methods
  - Sensitivity analysis
  - Design of experiments

- **Calibration**
  - Least squares calibration
  - Gaussian calibration
  - Bayesian calibration

- **Numerical methods**
  - Optimization
  - Integration
  - Least squares
  - Meshing
  - Coupling with external codes

OpenTURNS: documentation

- ▶ 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

▶ C++ and Python interface

▶ Parallel computations 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

▶
```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)
```

Coupling OpenTURNS with computer codes

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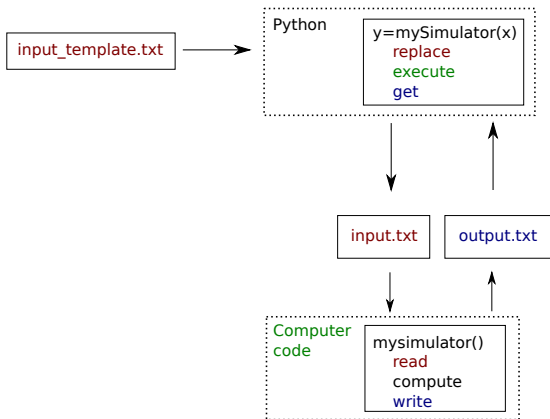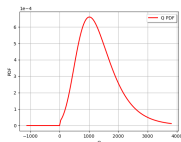

▶ Replaces the need for input/output text parsers

▶ 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

Probabilistic modeling
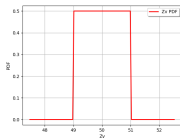
Random variables distributions:

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


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

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

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


```
d = ot.Normal(30.,7.5)
tr = ot.TruncatedDistribution.LOWER
Ks = ot.TruncatedDistribution(d, 0.0, tr)
```
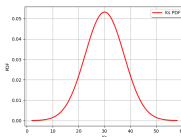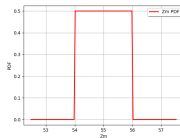
Zm: Uniform(min=54, max=56)

Probabilistic modeling

▶ 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
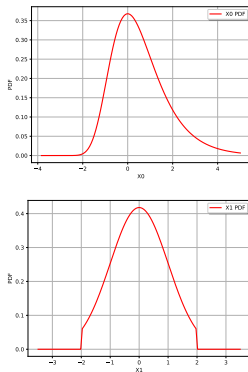
Joint distributions
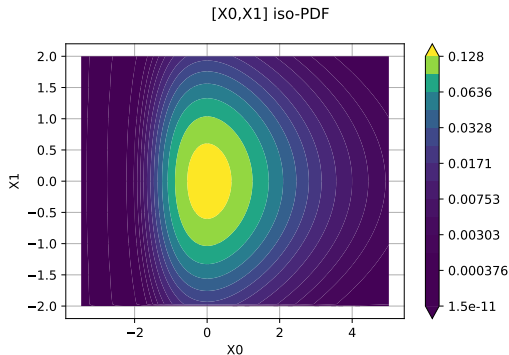
Beyond independent marginals: Copulas
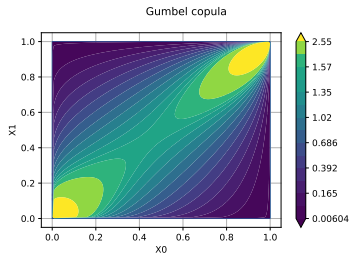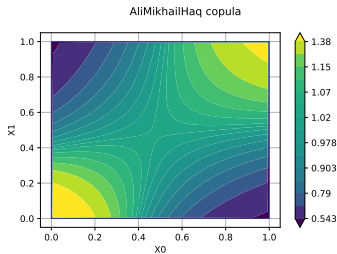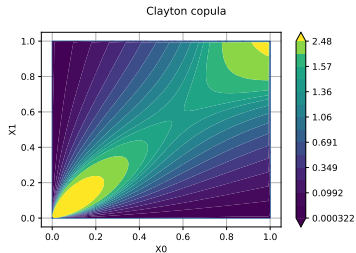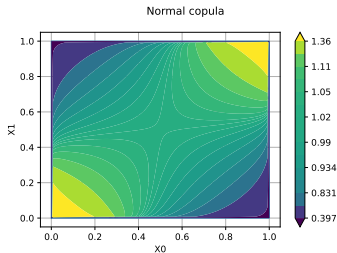
Composing marginal distributions and copulas

[X0,X1] iso-PDF

Gumbel copula

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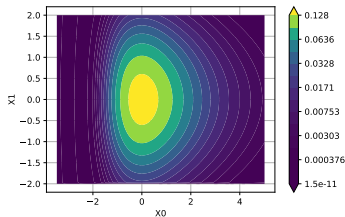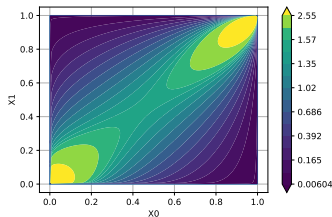
Joint distribution from marginals and copula
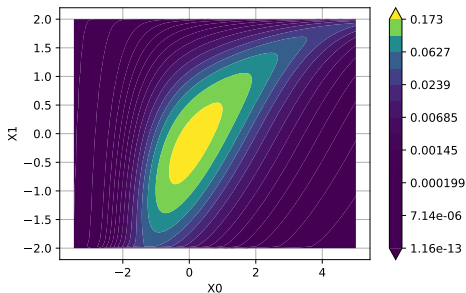
```
marg1 = ot.Gumbel(1.0, 0.0)
marg2 = ot.TruncatedNormal(0.0, 1.0, -2.0, 2.0)
copula = ot.GumbelCopula()
joint = ot.JointDistribution([marg1, marg2],
    copula)
graph = joint.drawPDF()
title = "Joint distribution "
title += "from marginals and copula"
graph.setTitle(title)
```

Design of experiments
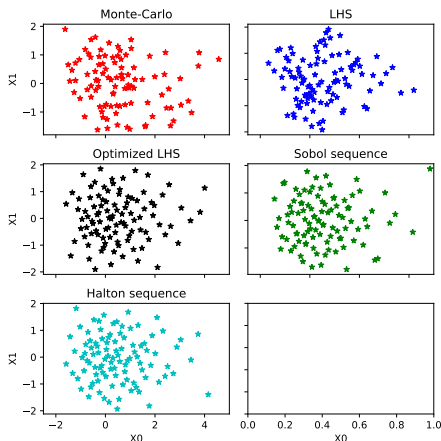

```
dim = 2
X=[ot.Gumbel(),ot.TruncatedNormal(0,1,-2,2)]
dist = ot.JointDistribution(X)
bounds = dist.getRange()
sampleSize = 100

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

Inference of the distribution $S = G(Q, K_s, Z_v, Z_m)$
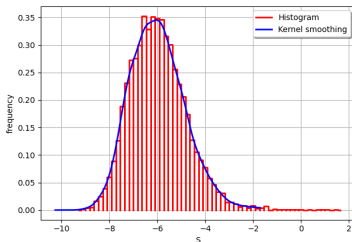

```python
Distribution = ot.JointDistribution([Q,Ks,Zv,Zm])

#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()
loiKS = ot.KernelSmoothing().build(sampleY)
graph2 = loiKS.drawPDF()
```

Distribution and dependence inference

Inference of the distribution $S = G(Q, K_s, Z_v, Z_m)$
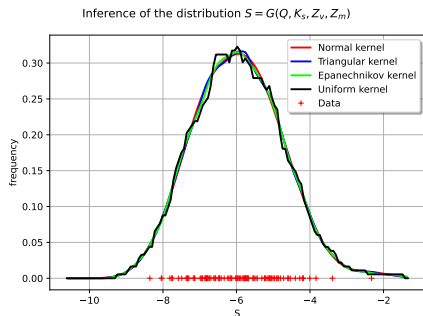

```
size = 100
sampleY = outputVector.getSample(
    size)
graph = ot.KernelSmoothing(ot.
    Normal()).build(sampleY).
    drawPDF()
loiKS = ot.KernelSmoothing(ot.
    Triangular()).build(sampleY)
graph2 = loiKS.drawPDF()
graph.add(graph2)
loiKS = ot.KernelSmoothing(ot.
    Epanechnikov()).build(sampleY)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

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

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

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

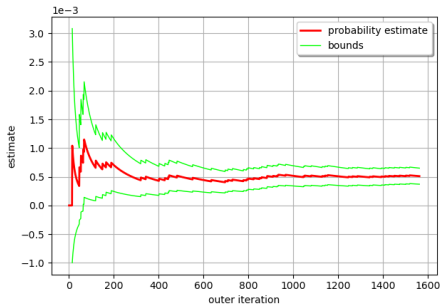Expectation convergence graph at level 0.95
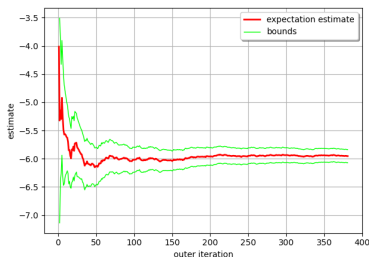
```
algo = ot.ExpectationSimulationAlgorithm(
    outputVector)
algo.setMaximumOuterSampling(100000)
algo.setBlockSize(1)
algo.setCoefficientOfVariationCriterionType("MAX")
algo.setMaximumCoefficientOfVariation(0.01)
algo.run()
graph = algo.drawExpectationConvergence()
```

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

$$\widehat{p_y} = \frac{1}{N}\sum_{1}^{N}\mathbb{I}_{G(\mathbf{x}_i)>0}$$

$$\widehat{\sigma} = \sqrt{\frac{1}{N}\sum_{1}^{N}(\mathbb{I}_{G(\mathbf{x}_i)>0} - \hat{p}_y)^2}$$

$$\widehat{\sigma_{p_y}} = \widehat{\sigma}/\sqrt{N}$$

```
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
    - ▶ Sobol' indices (multiple estimators)
    - ▶ FAST indices
    - ▶ ANCOVA indices
    - ▶ HSIC indices
    - ▶ Shapley Indices (available as a module)

Aggregated Sobol' indices - SaltelliSensitivityAlgorithm
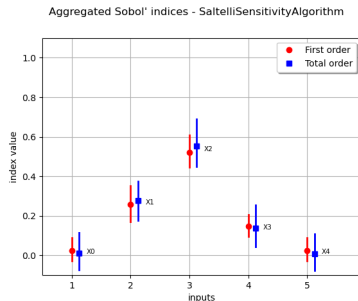
Sensitivity analysis: Parallel coordinates plot

Cobweb graph - [Y] vs [X1,X2,X3]
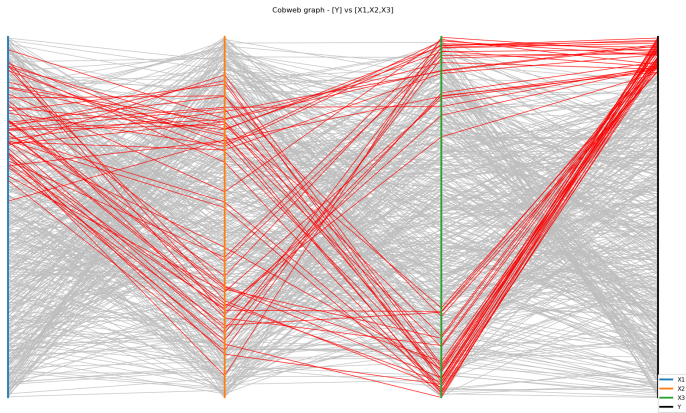
Sensitivity analysis: HSIC indices and associated p-values
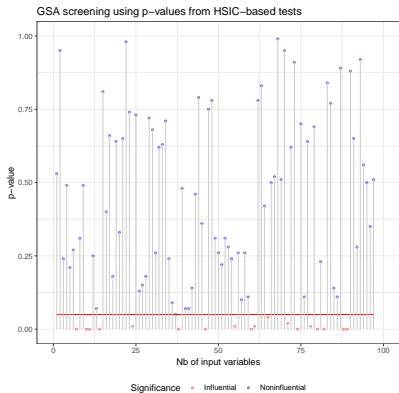
GSA-oriented screening.

TSA-oriented screening.

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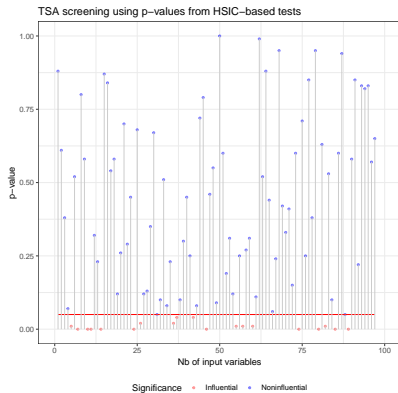
► Different surrogate modeling methods are available
  ► Kriging
  ► Polynomial chaos expansion
  ► Linear regression & step-wise basis selection
  ► Low rank tensors
  ► automatic validation tools

► 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
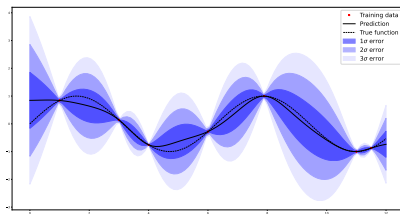

```python
inputSample = Distribution.getSample(100)
outputSample = fun(inputSample)

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
- ▶ Bounded and unbounded optimization
- ▶ Single and multi-objective optimization
- ▶ Multi-start wrapper

Field function modeling

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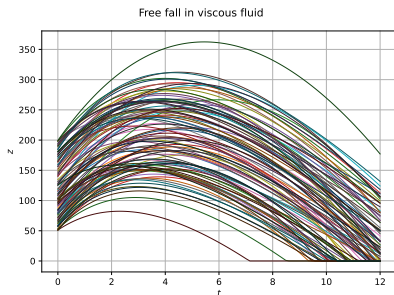
```python
def free_fall(X):
    g   = 9.81
    z0,v0,m,c = X
    tau=m/c
    vinf=-m*g/c
    t = np.array(mesh.getVertices().asPoint())
    z=z0+vinf*t+tau*(v0-vinf)*(1-np.exp(-t/tau))
    z=np.maximum(z,0.0)
    return ot.Field(mesh, z.reshape(-1, 1))

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)
```
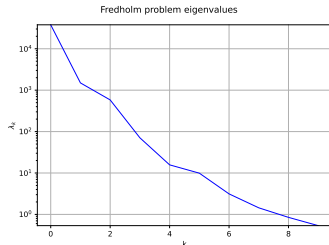
Dimension reduction: Karhunen-Loeve decomposition

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

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

Fredholm problem eigenvalues

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

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

eigenValues = KLResult.getEigenvalues()
```
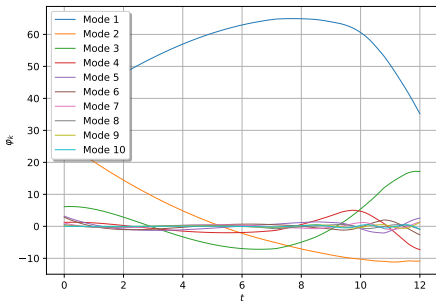
Dimension reduction: Karhunen-Loeve decomposition

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

Main modes:

Modes de KL, chute visqueuse

```
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

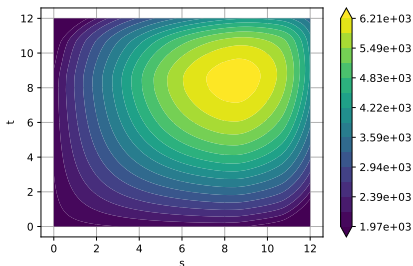We only consider the first 2 terms of the decomposition:

Trajectoires in the reduced space
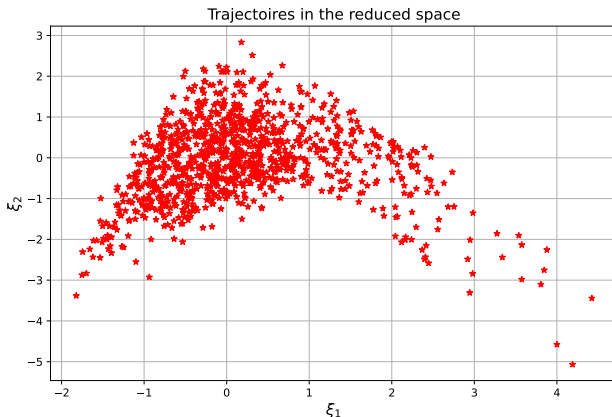
```
projectionFunction = ot.KarhunenLoeveProjection(KLResult)
sampleKsi = projectionFunction(outputFieldCentered)
sampleKsi = sampleKsi[:,:2]
```
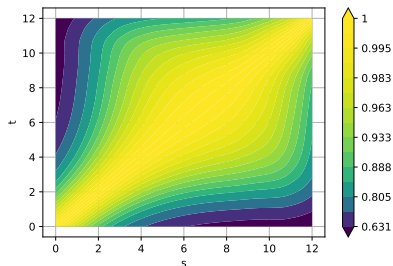
Field function analysis

We center the trajectories with respect to the mean field:

Viscous free fall covariance

Viscous free fall correlation

```
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

OpenTURNS Overview

Persalys Overview

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 $\rightarrow$ .AppImage (600 Mo)
    - ▶ Windows $\rightarrow$ .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"
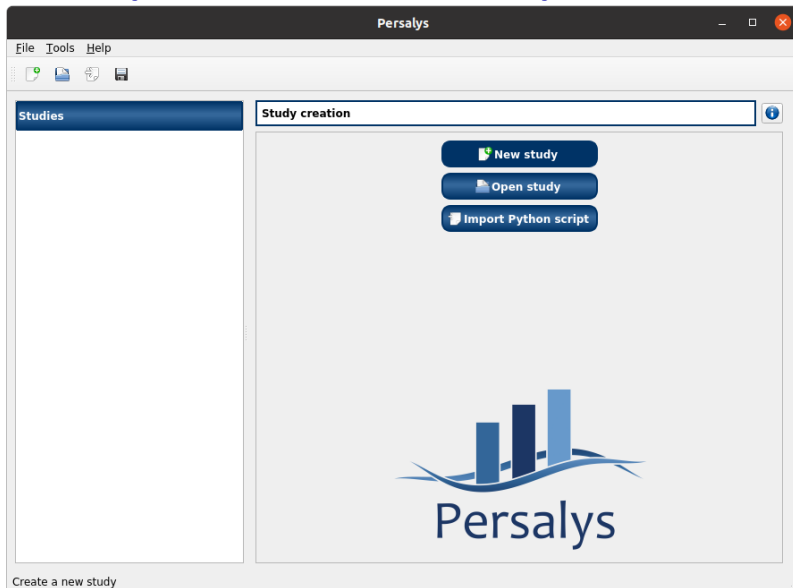
Create a study
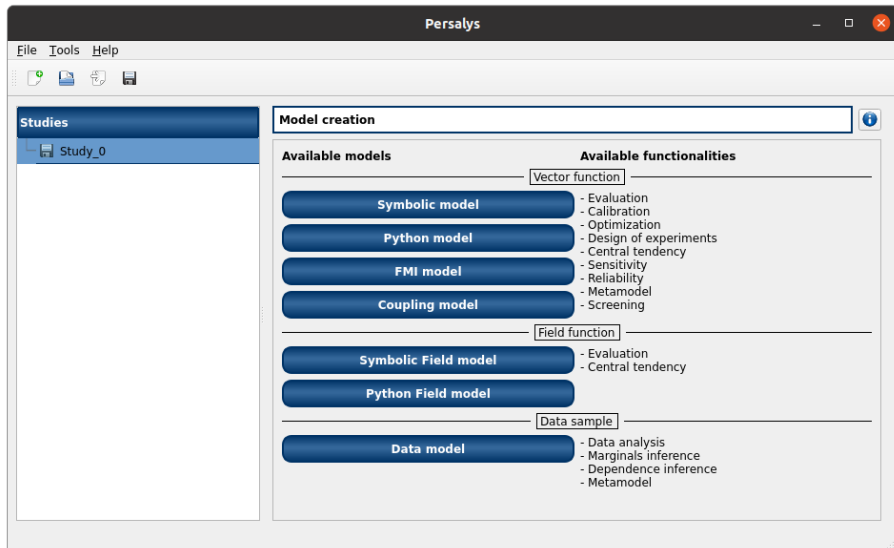
Definition/Evaluation

Models are viewed as a "black box" with inputs, outputs and a transfer function (TF). Persalys supports two model categories:

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

Vector to vector models can be of the following type:

- ▶ Symbolic, TF is a mathematical formula
- ▶ Python, TF is a Python function
- ▶ FMI, TF is provided by an FMU model
- ▶ Coupling, TF is an executable command which reads/writes input/output files
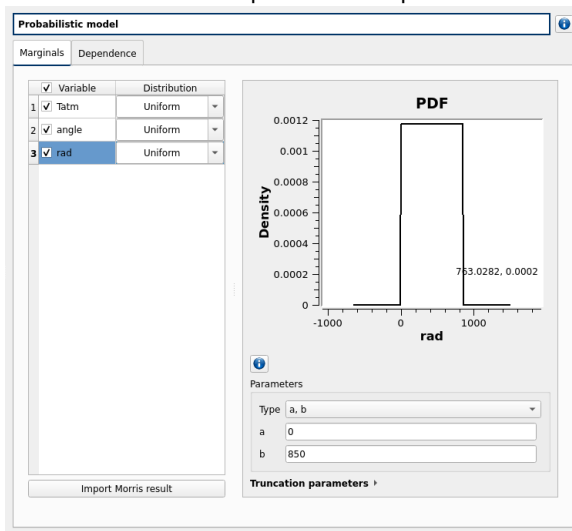
Coupling model definition

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.

Metamodel (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 being made.
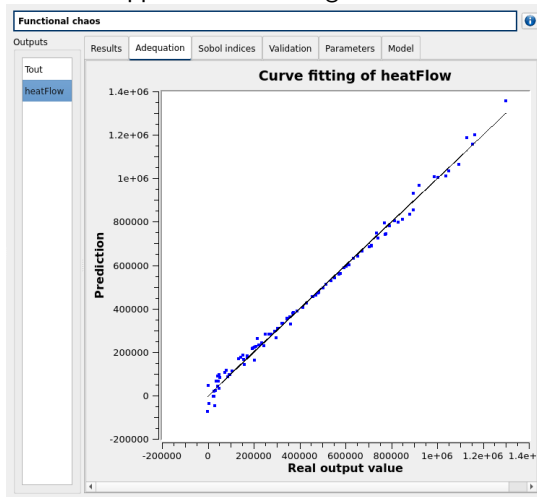
OpenTURNS resources

- ▶ 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`
- ▶ Forum: `https://persalys.discourse.group`