Overview of OpenTURNS and its graphical user interface

J. Pelamatti $^1$, M. Baudin $^1$, T. Delage $^1$

$^1$EDF R&D. 6, quai Watier, 78401, Chatou Cedex - France, julien.pelamatti@edf.fr

Journées du réseau MEXICO, 29-30 Novembre 2021, INRAE
Contents

Introduction

A few OpenTURNS functionalities and Examples

PERSALYS, the graphical user interface
OpenTURNS: www.openturns.org

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

- C++ library with a Python interface
- Numerical tools dedicated to the treatment of uncertainties
- Generic coupling to any type of physical model
- Open source, LGPL licensed

OpenTURNS: www.openturns.org

- Linux, Windows
- First release: 2007
- 5 full time developers
- 370,000 Total Conda downloads
- Project size (2018): 720 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 distributions
  - Copulas
  - Processes
  - Covariance models

- **Meta modeling**
  - 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

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

LevelSetMesher

Mesh of a level set

```python
levelSetMesher(wg)
```

Available constructor:
- `LevelSetMesher(discretization)`

Parameters:
- `discretization`: sequence of int, of dimension \( \leq 3 \)
  - 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 points problems. Default is `DefaultReferenceAlgorithm`.

Note
- The mesher 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 projection is true, then the projection is refined using an optimization solver.

Examples
- Create a mesh:
  ```python
  import openturns as ot
  mesh = ot.LevelSetMesher([10, 10])
  level = 1.0
  function = ot.SymbolicFunction(['x0', 'x1'], ['x0'^2 + 'x1'])
  levelFunction = ot.LevelFunction(function, level)
  mesh = mesh.build(levelFunction)
  ```

Methods
- `build(wg)`: Build the mesh of level set type.
- `getLevel()`: Accessor to the level.
- `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: modules

- Different python modules based on the OpenTURNS core classes
- Developed and maintained by the consortium
- Coded either in C++ or directly in Python

Main modules

- **otagram**: create a distribution from a Bayesian Network using aGrUM
- **otfftw**: Fast Fourier Transform algorithm (e.g. for stochastic processes) using FFTW
- **otfmi**: FMI models manipulation using PyFMI
- **otmixmod**: build mixtures of a multivariate Normal distribution from a sample
- **otmorris**: Morris screening method module
- **otpmml**: manages PMML files for meta-modeling exchanges
- **otpod**: A module to build Probability of Detection for Non Destructive Testing
- **otrobopt**: robust optimization
- **otsubsetinverse**: inverse subset simulation
- **otsvm**: Support Vector regression and classification with libsvm
- **otwrapy**: Python wrapper tools

Additional modules

- **otbenchmark**: benchmark problems for reliability and sensitivity analysis
- **othdrplot**: high density region algorithm for functional outlier detection
- **otsurrogate**: surrogate metamodels
- **otsklearn**: metamodels with the scikit-learn estimator API (fit/predict)
- **otusecases**: use cases suitable for OpenTURNS (functions and datasets)
- **otmarkov**: simulates Markov chains (experimental)
- **otsensitivity**: sensitivity analysis with density based measures
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 and source code
Example: Flood Analysis test-case

We consider the following random parameters:

- $Q$: Rate of flow ($m^3/s$)
- $K_s$: Strickler’s coefficient ($m^{1/3}/s$)
- $Z_v/Z_m$: Downstream/Upstream elevation (m)

Additionally, we consider the following parameters:

- River section length $L = 5000$ (m)
- River Width $B = 300$ (m) / Dam height $H_d = 58.5$

We approximate:

$$\alpha = \frac{Z_m - Z_v}{L},$$

Therefore:

$$H = \left(\frac{Q}{K_s B \sqrt{\alpha}}\right)^{0.6},$$

We want to analyse the overflow:

$$S = H + Z_v - H_d$$
A few OpenTURNS functionalities and Examples

Probabilistic modeling

Random variables distributions:

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

\[ Q \sim \text{Gumbel}(558, 1013) \]

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

\[ Zv \sim \text{Uniform}(49, 51) \]

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

\[ Ks \sim \text{Normal}(30, 7.5) \]

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

\[ Zm \sim \text{Uniform}(54, 56) \]
Monte-Carlo sampling

- The input distribution and relative output value are evaluated 10000 times
- The output distribution can be inferred through histogram or kernel smoothing methods

```python
# 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 infer the output distribution
size = 10000
sampleY = outputVector.getSample(size)
graph = ot.HistogramFactory().build(sampleY).drawPDF()
loiKS = ot.KernelSmoothing().build(sampleY)
graph2 = loiKS.drawPDF()
graph.add(graph2)
graph.setTitle(r'Inference of the distribution $S=G(Q,K_s,Z_v,Z_m)$')
graph.setXTitle('S')
graph.setYTitle('frequency')
graph.setLegends(['Histogram', 'Kernel smoothing'])
view = View(graph)
```
Distribution inference

- 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

```python
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()
graph.add(graph2)
graph.add(ot.Cloud(sampleY, ot.Sample(100,1)))
graph.setTitle(r'Inference of the distribution $S=G(Q, K_s , Z_v , Z_m )$')
graph.setXTitle('S')
graph.setYTitle(' frequency ') 
graph.setLegends(['Normal kernel', 'Triangular kernel', 'Epanechnikov kernel', 'Uniform kernel', 'Data'])
graph.setColors(['red', 'blue', 'green', 'black'])
view = View(graph)
```
Iterative Monte-Carlo: Central tendency analysis

- The expected value and associated standard deviation are computed iteratively.
- Different stopping criteria can be used: e.g., variation coefficient = 0.01
- Batch computation can be used.

![Expectation convergence graph at level 0.95](image)

\[
\hat{m}_y = \frac{1}{N} \sum_{1}^{N} G(X_i)
\]

\[
\hat{\sigma}_y = \sqrt{\frac{1}{N} \sum_{1}^{N} (G(X_i) - \hat{m}_y)^2}
\]

\[
\hat{\sigma}_{my} = \frac{\hat{\sigma}_y}{\sqrt{N}}
\]

```python
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)
```
A few OpenTURNS functionalities and Examples

Iterative Monte-Carlo: Reliability analysis

▶ We now consider the probability of flooding: \( P(S > 0) \).

▶ Same as before, but the function \( \mathbb{1}_{G(X_i) > 0} \) is considered.

\[
\hat{p}_y = \frac{1}{N} \sum_{1}^{N} \mathbb{1}_{G(X_i) > 0}
\]

\[
\hat{\sigma} = \sqrt{\frac{1}{N} \sum_{1}^{N} (\mathbb{1}_{G(X_i) > 0} - \hat{p}_y)^2}
\]

\[
\hat{\sigma}_{p_y} = \frac{\hat{\sigma}}{\sqrt{N}}
\]

```python
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()
graph = algo.drawProbabilityConvergence()
view = View(graph)
```

OpenTURNS
A few OpenTURNS functionalities and Examples

FORM/SORM reliability analysis

- We estimate the probability of flooding through FORM/SORM procedures
- MC estimation requires $\sim 1500$ function evaluations
- FORM and SORM only use $\sim 150$

Estimated probability:
- MC: $5.09999999999998 \times 10^{-4}$
- FORM: $5.340929030055227 \times 10^{-4}$
- SORM: $6.793780433482759 \times 10^{-4}$

Also:
- Directional sampling
- Importance sampling
- Subset sampling

Different types and parameterizations of finite difference gradient computation are available
Sensitivity analysis : Sobol indices

- We want to identify the most influential random input parameters
- Different sensitivity analysis methods : Sobol, SRC, Morris
- We generate an appropriate design of experiment and use it to compute the Sobol indices of the 4 inputs
- Different estimators are available

\[
S_i = \frac{\text{Var}[\mathbb{E}[Y|X_i]]}{\text{Var}[Y]}
\]

\begin{verbatim}
size = 1000
computeSecondOrder = False
sie = ot.SobolIndicesExperiment(Distribution, size, computeSecondOrder)
inputDesign = sie.generate()
outputDesign = fun(inputDesign)
sensitivityAnalysis = ot.
    SaltelliSensitivityAlgorithm(
        inputDesign, outputDesign, size)
sensitivityAnalysis.setBootstrapSize(300)
graph = sensitivityAnalysis.draw()
view = View(graph)
\end{verbatim}
Sensitivity analysis : HSIC indices

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

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

$$
\mathcal{M} : \begin{array}{c}
\mathcal{X} \subseteq \mathbb{R}^d \\
\mathbf{X} \mapsto \mathbf{Y} = \mathcal{M}(\mathbf{X})
\end{array}
$$

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

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

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

Let us now consider $\mathcal{V}$, an RKHS over $\mathcal{X} \times \mathcal{Y}$ with kernel $\nu(\cdot, \cdot)$.

We consider the mean embedding of $P_{\mathcal{Y}} P_{\mathcal{X}_i}$ and $P_{\mathcal{Y}, \mathcal{X}_i}$ in $\mathcal{V}$.

$$\mu[P_{\mathcal{Y}} P_{\mathcal{X}_i}] = \mathbb{E}_Y \mathbb{E}_{X_i}[\nu((Y, X_i), \cdot)]$$

$$\mu[P_{\mathcal{Y}, \mathcal{X}_i}] = \mathbb{E}_{Y, X_i}[\nu((Y, X_i), \cdot)]$$

We now have a dependence measure under the form of:

$$\Delta := ||\mu[P_{\mathcal{Y}, \mathcal{X}_i}] - \mu[P_{\mathcal{Y}} P_{\mathcal{X}_i}]|| \longleftrightarrow \text{HSIC}(Y, X_i) = \Delta^2$$
Sensitivity analysis : HSIC indices

Estimators

- Different estimators, e.g., V-statistics
  \[
  \hat{\text{HSIC}}(X_i, Y) = \frac{1}{n^2} \text{Tr}(L_iHLH)
  \]
  where \( L_i \) and \( L \) are Gram matrices, and \( H \) a shift matrix
  → Kernel-based estimator

- A plug-in estimator of a normalized sensitivity index
  \[
  R_{\text{HSIC},i}^2 = \frac{\hat{\text{HSIC}}(X_i, Y)}{\sqrt{\hat{\text{HSIC}}(X_i, X_i) \hat{\text{HSIC}}(Y, Y)}}
  \]  (1)
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!
Sensitivity analysis: HSIC indices

Statistical hypothesis testing with HSIC

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

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

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

- $\hat{S}_T := n \times \hat{\text{HSIC}}(X_i, Y)$ is the test statistic

- **p-value** associated to the test $\mathcal{T}$:

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

- Asymptotic estimator (small data set)
- Permutation-based estimator (large data set)
Sensitivity analysis : HSIC indices

- Illustrative scheme of an IBLOCA scenario (©CEA)
- Simulation trajectories of the Peak Cladding Temperature (PCT) (©EDF).
Sensitivity analysis: HSIC indices

GSA screening using p-values from HSIC-based tests

TSA screening using p-values from HSIC-based tests

GSA-oriented screening.

TSA-oriented screening.

Screening for GSA & TSA using p-values.
Sensitivity analysis : HSIC indices

GSA-oriented ranking

TSA-oriented ranking

Ranking for GSA & TSA using p-values and $R^2_{HSIC}$
Sensitivity analysis: HSIC indices

Work in progress!

Coming soon in OpenTURNS 1.19 (~ May 2022)
Surrogate modeling : Kriging

- Different surrogate modeling methods are available
  - Kriging
  - Polynomial chaos expansion
  - Linear regression
  - Low rank tensors

- Kriging
  - 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()
```
Surrogate modeling : Validation

- Automated validation of surrogate models with respect to a user provided data set
- Predictivity factor: $Q^2 = 0.992615$
- $y - \hat{y}$ plot:

```python
inputValidation = Distribution.getSample(1000)
outputValidation = fun(inputValidation)
validation = ot.MetaModelValidation(
    inputValidation, outputValidation, KrigingMM)
graph = validation.drawValidation()
view = View(graph)
print(validation.computePredictivityFactor())
```
Optimization

- OpenTURNS provides an interface with several optimization libraries
  - AUGLAG
  - AUGLAG_EQ
  - GD_MLSL
  - GD_MLSL_LDS
  - GD_STOGO
  - GD_STOGO_RAND
  - GN_AGS
  - GN_CRS2_LM
  - GN_DIRECT
  - GN_DIRECT_L
  - GN_DIRECT_L_NOSCAL
  - GN_DIRECT_L_RAND
  - GN_DIRECT_L_RAND
  - GN_DIRECT_NOSCAL
  - GN_ESCH, GN_ISRES
  - GN_MLSL
  - GN_MLSL_LDS
  - GN_ORIG_DIRECT
  - GN_ORIG_DIRECT_L
  - G_MLSL
  - G_MLSL_LDS
  - LD_AUGLAG
  - LD_AUGLAG_EQ
  - LD_CCSAQ
  - LD_LBFGS
  - LD_MMA
  - LD_SLSQP
  - LD_TNEWTON
  - LD_TNEWTON_PRECOND
  - LD_VAR1
  - LD_VAR2
  - LN_AUGLAG
  - LN_AUGLAG_EQ
  - LN_BOBYQA
  - LN_COBYLA
  - LN_NELDERMEAD
  - LN_NEWUOA
  - LN_NEWUOA_BOUND
  - LN_PRAXIS
  - LN_SBPLX

- Constrained and unconstrained optimization
- Bound and unbound optimization
- Multi-start wrapper
Optimization

We want to maximize the value of the overflow in order to identify the most dangerous configuration of input variables.

```
# Setting the optimization bounds and starting point distribution
lbounds = [1e-3, 1e-3, 49, 54]
ubounds = [4000, 60, 51, 56]
MSDistribution = []

for i in range(dimension):
    MSDistribution.append(ot.Uniform(lbounds[i], ubounds[i]))

MSDistribution = ot.ComposedDistribution(MSDistribution)
bounds = ot.Interval(ot.Point(lbounds), ot.Point(ubounds))

# Defining the optimization problem
problem = ot.OptimizationProblem(fun)
problem.setMinimization(False)
problem.setBounds(bounds)

# Defining and parameterizing the solver
algo = ot.MultiStart(ot.NLopt('LN_COBYLA'), MSDistribution.getSample(10))
 algo.setProblem(problem)
 algo.setMaximumRelativeError(1e-6)

# Running the optimization
algo.run()
```
Optimization

```python
res = algo.getResult()
OptValue = res.getOptimalValue()
OptPoint = res.getOptimalPoint()
graph = res.drawOptimalValueHistory()

print(OptValue)
[1.09192]

print(OptPoint)
[3638.55, 13.5605, 50.9784, 54.0316]

print(lbounds)
[69.78551963374207, 13.560460816677406, 49.02524785938163, 54.031603399244695]

print(ubounds)
[3638.549738548708, 48.154203046100534, 50.978367971379114, 55.94973015924193]
```

- As expected, we hit the min/max bounds in order to find the maximal value of overflow
Design of experiments

- Different Design of experiment types are available
- 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

[X0,X1] iso-PDF
A few OpenTURNS functionalities and Examples

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()

sequence = ot.HaltonSequence(dim)
experiment = ot.LowDiscrepancyExperiment(sequence, distribution, sampleSize, False)
```
Beyond independent marginals: Copulas

Gaussian copula

Ali Mikhail Haq copula

Clayton copula

Gumbel copula

OpenTURNS 33/63
Composing marginal distributions and copulas

We obtain:

```
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)
```

Different copulas can be used to link the various dimensions of the model
Example: Viscous free fall

- We consider here the free fall of a sphere in a viscous fluid.

- We model the vertical trajectory of the ball as a function of 4 random input parameters:
  - \( z_0 \): Initial height: Uniform( mean = 50.0, std = 200.0 )
  - \( v_0 \): Initial vertical speed: Normal( min = 55.0, max = 10.0 )
  - \( m \): Ball mass: Normal( mean = 80.0, std = 8.0 )
  - \( c \): Fluid viscosity: Uniform( min = 0.0, max = 30.0 )

- The model can be seen as a field function with the following expression:

\[
z(t) = z_0 + v_{\text{inf}} t + \tau (v_0 - v_{\text{inf}})(1 - e^{-t/\tau}) \quad \forall t \in [0, t_{\text{max}}]
\]

\[
\tau = \frac{m}{c}
\]

\[
v_{\text{inf}} = -m \times g \times c
\]

- If the model inputs are random, the output can be seen as a stochastic process

Field function sampling

```python
def AltiFunc(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)
Field function analysis

```python
meanField = outputField.computeMean()
graph = meanField.drawMarginal(0)
graph.setTitle('Free fall in viscous fluid')
graph.setXTitle(r'$t$')
graph.setYTitle(r'$z$')
quantileField_005 = outputField.computeQuantilePerComponent(0.05)
graph.add(quantileField_005.drawMarginal(0))
quantileField_095 = outputField.computeQuantilePerComponent(0.95)
graph.add(quantileField_095.drawMarginal(0))
quantileField_0005 = outputField.computeQuantilePerComponent(0.005)
graph.add(quantileField_0005.drawMarginal(0))
quantileField_0995 = outputField.computeQuantilePerComponent(0.995)
graph.add(quantileField_0995.drawMarginal(0))
```
Field function analysis

We center the trajectories with respect to the mean field:

```
meanFunction = P1LagrangeEvaluation(meanField)
trend = TrendTransform(meanFunction, myMesh)
invTrend = trend.getInverse()
outputFieldCentered = invTrend(outputField)
```
Dimension reduction : Karhunen-Loève decomposition

- We wish to reduce the dimension of the problem from an infinite dimensional output to a finite dimensional one
- We can perform a Karhunen-Loève 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

\[ Y(\omega, t) = \sum_{k=1}^{\infty} \sqrt{\lambda_k} \xi_k(\omega) \phi_k(t) \rightarrow \tilde{Y}(\omega, t) = \sum_{k=1}^{p} \sqrt{\lambda_k} \xi_k(\omega) \phi_k(t) \]

Fredholm problem eigenvalues

```python
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-Loève decomposition

\[ \tilde{Y}(\omega, t) = \sum_{k=1}^{p} \sqrt{\lambda_k} \xi_k(\omega) \varphi_k(t) \]

Main modes:
Dimension reduction: Karhunen-Loève decomposition

We only consider the first 2 terms of the decomposition:

```python
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)
view=View(graph)

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

We can combine the Karhunen-Loève decomposition and a polynomial chaos regression in order to predict new fields:

1. Train a projection function on a training data set: Field space $\rightarrow$ Reduced space
2. Define the lifting function: Reduced space $\rightarrow$ Field space
3. Train a polynomial chaos regressor: Input space $\rightarrow$ Reduced space
4. Generate a new sample in the Input space
5. Predict its image in the reduced space $\rightarrow$ Lift the result in the Field space

\[ X \xrightarrow{g} Y \]

\[ (\xi_k)_{k \in \mathbb{N}} \]

\[ g_{\xi} \]

\[ \pi \]

\[ (\pi')^{-1} \]
Field function surrogate model

# Creation of the orthonormal polynomial basis
enumerateFunction = ot.HyperbolicAnisotropicEnumerateFunction(distX.getDimension(), 0.7)
basis = ot.OrthogonalProductPolynomialFactory([ot.StandardDistributionPolynomialFactory(
    distX.getMarginal(i)) for i in range(distX.getDimension())])

# Creation of the basis selection strategy
degree = 7
adaptive = ot.FixedStrategy(basis, enumerateFunction.getStrataCumulatedCardinal(degree))

# Input space to Reduced space PCE
algo = FunctionalChaosAlgorithm(inputSample, sampleKsi, distX, adaptive)
algo.run()

polyChaos = algo.getResult().getMetaModel()
polyResults = algo.getResult()

# Creation of the Input to Field space surrogate model
procZ_centre = ot.PointToFieldConnection(liftFunction, polyChaos)
metaModel = ot.PointToFieldConnection(trend, procZ_centre)
Field function surrogate model

Comparaison model/surrogate model
Time dependent sensitivity analysis

The Polynomial chaos expansion provides the Sobol indices as a free byproduct of the surrogate model training. We can use this feature in order to compute the sensitivity indices as a function of time

```python
chaosSI = FunctionalChaosSobolIndices(polyResults)
chaosRV = FunctionalChaosRandomVector(polyResults)
Modes = KLResult.getModesAsProcessSample()

SIField = ProcessSample()

VarFieldValues = 0.
for i in range(eigenValues.getSize()):
    VarFieldValues = VarFieldValues + eigenValues[i] * (np.array(Modes.getMarginal(0)[i]) ** 2 * chaosRV.getCovariance()[:,i,i])

for j in range(distX.getDimension()):
    VarFieldValuesX = 0.
    for i in range(eigenValues.getSize()):
        VarFieldValuesX = VarFieldValuesX + eigenValues[i] * (np.array(Modes.getMarginal(0)[i]) ** 2 * chaosRV.getCovariance()[:,i,i] * chaosSI.getSobolTotalIndex(j,i))
    SIField.add(Field(myMesh, VarFieldValuesX / VarFieldValues))

graph = SIField.drawMarginal()
graph.setTitle(r'Viscous free fall sensitivity indices')
graph.setXTitle(r'$t$')
graph.setYTitle(r'$SI_{total}$')
graph.setLegends(distX.getDescription())
view = View(graph)
```
Time dependent sensitivity analysis

Viscous free fall sensitivity indices

- Zo
- Vo
- M
- C

Graph showing the sensitivity indices over time.
Calibration of a computer code

- The objective is to determine the optimal parameters of a computer code by using the available data
- We consider the following model:

\[ y = \theta_0 + x\theta_1 + x^2\theta_2 + \varepsilon \]

- True values: \( \theta = \{-4.5, 4.8, 2.2\} \)
- 10 observations:

![Graph showing y0 as a function of x1](image)

- Linear least squares
- Non-linear least squares
- Gaussian calibration
- Bayesian calibration
Bayesian calibration

- We consider a multivariate Gaussian prior on $\theta$:

$$\theta \sim \mathcal{N} \left( \mu \begin{bmatrix} -3 \\ 4 \\ 1 \end{bmatrix}, \sigma = \begin{bmatrix} 2^2 & 0 & 0 \\ 0 & 1^2 & 0 \\ 0 & 0 & 1.5^2 \end{bmatrix} \right)$$

- We can then use a MCMC to sample from the conditioned posterior distribution (1000 samples + Kernel smoothing):
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**: 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
Graphical interface of OpenTURNS

Features:
- Probabilistic modeling
- Distribution fitting
- Central tendency
- Sensitivity analysis
- Probability estimate
- Meta-modeling (polynomial chaos, kriging)
- Screening (Morris)
- Optimization
- Design of experiments
- 1-D field analysis

Partners: EDF, Phiméca

Licence: LGPL

persalys.fr
salome-platform.org
PERSALYS: model definition

Different options for defining the model:

- Analytical formula
- Python script
- Numerical code called through a file exchange system
- FMU models
- Data import
The probabilistic model can be defined in different ways:

- Independent parametric marginals
- Marginal coupled through copulas
- Inference of marginals from data
- Inference of copulas from data
PERSALYS, the graphical user interface

PERSALYS: data analysis

- Parametric and non-parametric distribution inference
PERSALYS, the graphical user interface

PERSALYS: data analysis

Visual data analysis:

- Pair-wise scatter plots
- Empirical histogram
- Interactive and linked views
- Physical and rank spaces
PERSALYS, the graphical user interface

PERSALYS: data analysis

Interactive graphical data analysis
PERSALYS: 1D fields

- Mesh definition and visualization
- Import from text or csv file
PERSALYS, the graphical user interface

PERSALYS: 1D fields

- Functional model definition and probabilistic model
- Python or symbolic

Python model

```python
from numpy import maximum, exp
def exec(z0, v0, m, c):
g = 9.81
zmin = 0.
tau = m / c
vinf = -m * g / c
# mesh nodes
t = getMesh().getVertices()
z = [maximum(z0 + vinf * t_i[0] + tau)
return z
```

<table>
<thead>
<tr>
<th>Index parameter: t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
</tr>
<tr>
<td>✓ Name</td>
</tr>
<tr>
<td>1  z0</td>
</tr>
<tr>
<td>2  v0</td>
</tr>
<tr>
<td>3  m</td>
</tr>
<tr>
<td>4  c</td>
</tr>
</tbody>
</table>

Graph settings

- Title: Trajectory
- Data: Select
- X-axis: Y-axis: Plot style
- Title: t
- Min: 0
- Max: 12
- Log scale

Trajectory
PERSALYS, the graphical user interface

PERSALYS: 1D fields

- Probabilistic model
- Uncertainty propagation with simple Monte-Carlo sampling
PERSALYS: 1D fields

- BagChart and Functional Bagchart (from Paraview) based on High Density Regions (Hyndman, 1996).
- To do this, Paraview uses a principal component analysis decomposition.
- Linked and interactive selections in the views.

Support, discussion and contribution

- **github repository**: https://github.com/persalys/persalys
  - Bug report
  - Enhancement suggestions
  - Mirror of the internal development git repository

- **Discourse forum**: https://persalys.discourse.group/
  - Practical questions
  - Theoretical questions
  - Feature request
  - Forum layout
Thank you for your attention! :)