Overview of OpenTURNS, its new features 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

March 1st 2024, SIAM UQ 2024, Trieste, Italy

Contents

OpenTURNS: www.openturns.org

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

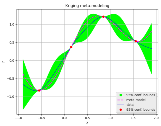
- Multivariate probabilistic modeling including dependence
- Numerical tools dedicated to the treatment of uncertainties
- Generic coupling to any type of physical model
- Open source, LGPL licensed, C++/Python library

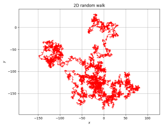
OpenTURNS: www.openturns.org

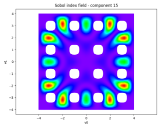
- Linux, Windows, macOS
- First release : 2007
- 5 full time developers
- Users $\approx$ 1000, mainly in France (1 078 000 Total Conda downloads)
- Project size : 800 classes, more than 6000 services

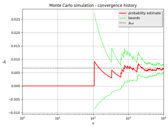
OpenTURNS: content

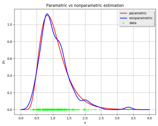
- Data analysis
  - Distribution fitting
  - Statistical tests
  - Estimate dependency and copulas
  - Estimate stochastic processes
- Probabilistic modeling
  - Dependence modeling
  - Univariate distributions
  - Multivariate distrbutions
  - Copulas
  - Processes
  - Covariance kernels
- Surrogate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







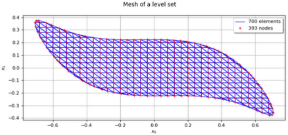




OpenTURNS: documentation

LevelSetMesher

(Source code, png, hires.png, pdf)



class **LevelSetMesher**(**args*)

Creation of mesh of box type.

Available constructor:

LevelSetMesher(*discretization*)

Parameters: **discretization** : sequence of int, of dimension ≤ 3.

Discretization of the levelset bounding box.

solver : OptimizationAlgorithm

Optimization solver used to project the vertices onto the level set. It must be able to solve nearest point problems. Default is AbdoRackwitz.

Notes

The meshing algorithm is based on the IntervalMesher class. First, the bounding box of the level set (provided by the user or automatically computed) is meshed. Then, all the simplices with all vertices outside of the level set are rejected, while the simplices with all vertices inside of the level set are kept. The remaining simplices are adapted the following way :

- The mean point of the vertices inside of the level set is computed
- Each vertex outside of the level set is projected onto the levelset using a linear interpolation
- If the *project* flag is *True*, then the projection is refined using an optimization solver.

Examples

Create a mesh:

```
>>> import openturns as ot
>>> mesher = ot.LevelSetMesher([5, 10])
>>> level = 1.0
>>> function = ot.SymbolicFunction(['x0', 'x1'], ['x0^2+x1^2'])
>>> levelSet = ot.LevelSet(function, level)
>>> mesh = mesher.build(levelSet)
```

Methods

build(*args)	Build the mesh of levelset type.
getClassName()	Accessor to the object's name.
getDiscretization()	Accessor to the discretization.

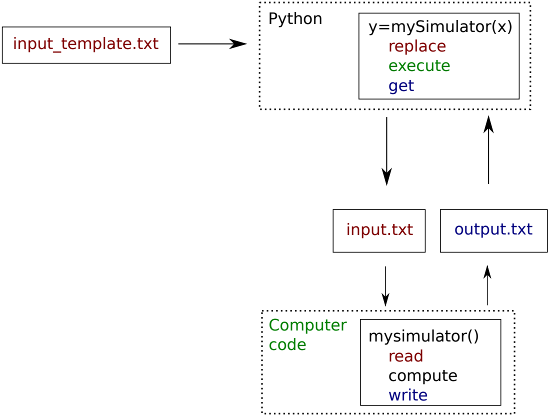
- Content:
  - Programming interface (API)
  - Examples
  - Theory
- *All* classes and methods are documented, partly automatically.
- Examples are automatically tested at *each* update of the code and outputs are checked.

OpenTURNS: practical use

- 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

Coupling OpenTURNS with computer codes

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



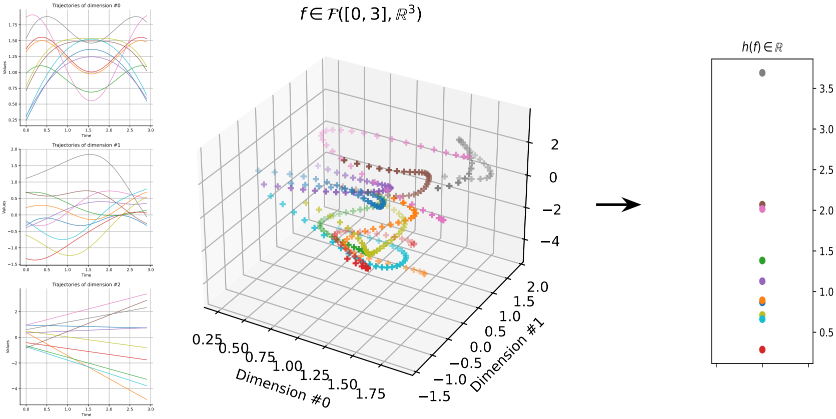
- Replaces the need for input/output text parsers
- Wraps a simulation code under the form of a standard python function
- Allows to interface OpenTURNS with a cluster
- otwrapy: interfacing tool to allow easy parallelization

Contents

Functional surrogate model

- We want a surrogate $\hat{h}$ of a model h that maps a time series to a vector:

$$h : \mathcal{F}([0,1], \mathbb{R}^d) \to \mathbb{R}^p$$



Functional surrogate model: FieldToPointFunctionalChaosAlgorithm

- We build the surrogate $\hat{h} : \mathcal{F}([0,1], \mathbb{R}^d) \to \mathbb{R}^p$ from N observations.
- Define the observations:

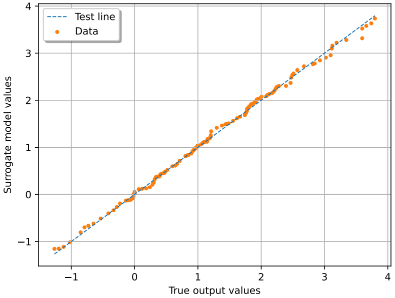
```
import openturns as ot
timegrid = ot.RegularGrid(start, step, NT) # NT is the number of time steps
collection = ... # collection of N numpy arrays with shape (NT, d)
x = ot.ProcessSample(timegrid, collection) # functional input data
y = ... # numpy array of shape (N, p)
```

- Step 1: for each dimension of the input data, perform PCA.
- Step 2: build a polynomial chaos mapping the PCA coefficients to the outputs
- The `FieldToPointFunctionalChaosAlgorithm` experimental class does both steps at once:

```
from openturns.experimental import FieldToPointFunctionalChaosAlgorithm
algo = FieldToPointFunctionalChaosAlgorithm(x, y)
algo.setThreshold(0.04) # part of the variance unexplained by the PCA
algo.run()
result = algo.getResult()
```

Functional surrogate model: validation (case where $p = 1$)

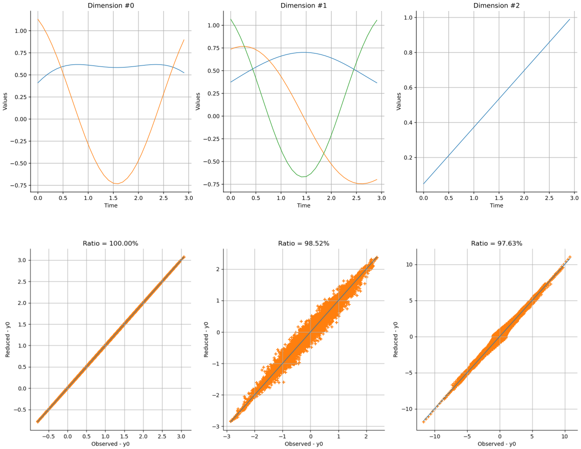
```
x_valid = ... # input validation ProcessSample
y_valid = ... # output values
surrogate = result.getFieldToPointMetamodel() # h_hat
yhat_valid = surrogate(x_valid)

graph = ot.VisualTest.DrawQQplot(y_valid, yhat_valid) # graphical validation
graph.setTitle("")
graph.setXTitle("True output values")
graph.setYTitle("Surrogate model values")
```



Step 1 analysis: visualize and validate PCA modes

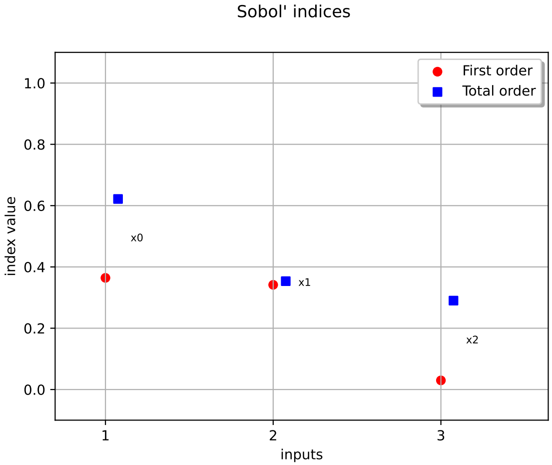
```
pca_l = result.getInputKLResultCollection()
grid = ot.GridLayout(1, d)
for num, pca in enumerate(pca_l):
    modes = pca.getModesAsProcessSample()
    graph = modes.drawMarginal(0)
    graph.setTitle(f"Dimension #{num}")
    grid.setGraph(0, num, graph)
```

```
for num, pca in enumerate(pca_l):
    m = x.getMarginal(num)
    v = ot.KarhunenLoeveValidation(m, pca)
    graph= v.drawValidation().getGraph(0,0)
    ratio = 100.0 * pca.getSelectionRatio()
    graph.setTitle(f"Ratio = {ratio:.2f}%")
    grid.setGraph(0, num, graph)
```



Step 2 analysis: polynomial-chaos-derived Sobol' indices

```
frop openturns.experimental import FieldFunctionalChaosSobolIndices
sensitivity = FieldFunctionalChaosSobolIndices(result)
graph = sensitivity.draw()
```

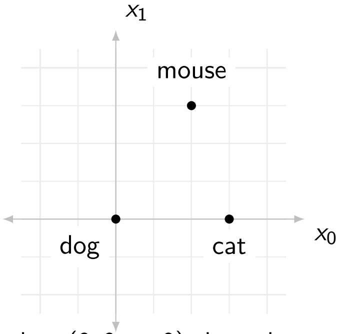


Contents

Surrogate with mixed continuous and categorical variables

Mapping categories to a latent space with Gaussian correlation kernel[1]

- dog
- cat
- mouse



- First category: automatically mapped to $(0, 0, ..., 0)$: here dog to $(0, 0)$.
- Second category: automatically mapped to $(x, 0, ..., 0)$: here cat to $(x, 0)$.

```python
from openturns.experimental import LatentVariableModel
covModel = LatentVariableModel(3, 2) # 3 categories (dog, cat, mouse) in 2D
activeCoordinates = [1.5, 1.0, 1.5] # [cat_0, mouse_0, mouse_1]
covModel.setLatentVariables(activeCoordinates)
```

[1] Zhang, Y., Tao, S., Chen, W., and Apley, D. W. (2020)

Example: 1 continuous + 1 boolean variable

Product of a 5/2 Matérn covariance kernel for the continuous variable and a latent variable kernel for the boolean variable:

```
# Standard 5/2 Matern covariance kernel for the continuous variable
kx = ot.MaternModel()
kx.setNu(2.5) # smoothess

# Latent variable kernel for the boolean variable
from openturns.experimental import LatentVariableModel
kz = LatentVariableModel(2, 1) # 2 categories in a 1D latent space

# Mixed variable kernel: product of the two kernels
kLV = ot.ProductCovarianceModel([kx, kz])
```

Set the bounds for the optimization of the kernel parameters:

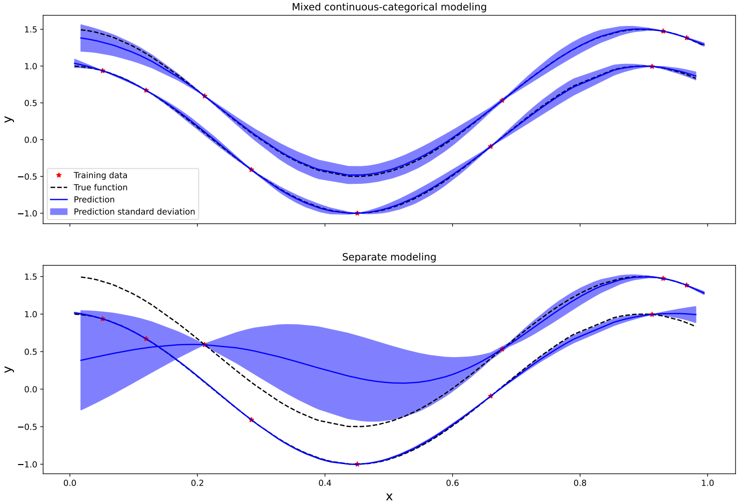
```
# The latent variable kernel amplitude is set to 1: it is fixed.
param_number = kLV.getParameter().getSize() - 1 # -1 because kz amplitude is fixed
lowerBoundLV = [1e-4] * param_number
upperBoundLV = [2.0] * param_number
boundsLV = ot.Interval(lowerBoundLV, upperBoundLV) # param_number-dimensional interval
```

Sample points to initialize the optimization algorithm:

```
unif_coll = [ot.Uniform(lowerBoundLV[i], upperBoundLV[i]) for i in range(param_number)]
initDistLV = ot.ComposedDistribution(unif_coll) # uniform distribution in boundsLV
initSampleLV = initDistLV.getSample(30) # a sample from this distribution to start ...
optAlgLV = ot.MultiStart(ot.NLopt("LN_COBYLA"), initSampleLV) # ... the optimization
```

Example – continued: Gaussian Process Regression

```
algoLV = ot.KrigingAlgorithm(x, y, kLV) # x: input, y: output
algoLV.setOptimizationAlgorithm(optAlgLV)
algoLV.setOptimizationBounds(boundsLV)
algoLV.run()
```



Contents

Project overview

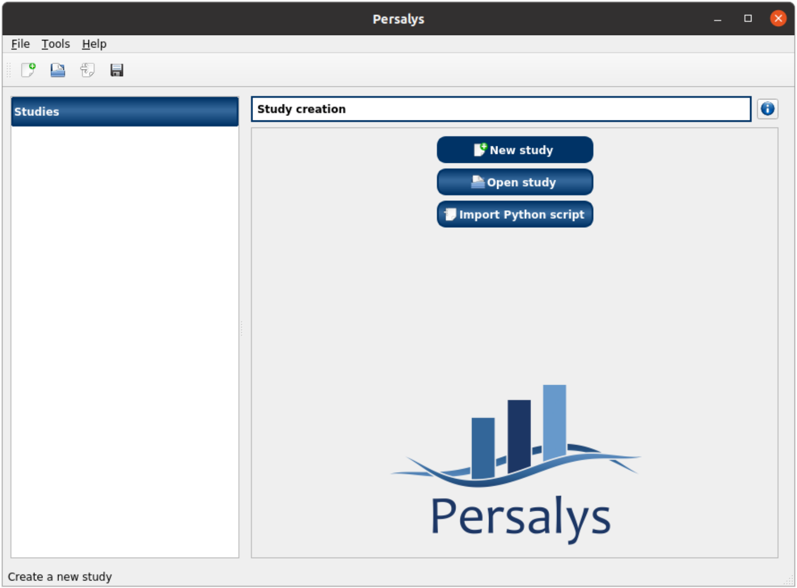
- Partnership between EDF and Phimeca since 2015
  - Developed in C++ using Qt
  - Aiming at maximizing the use of OpenTURNS - through a dedicated GUI - for engineers/researchers without a strong coding experience
  - As easy to use as possible while providing the user with help and guidelines
  - Benefit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

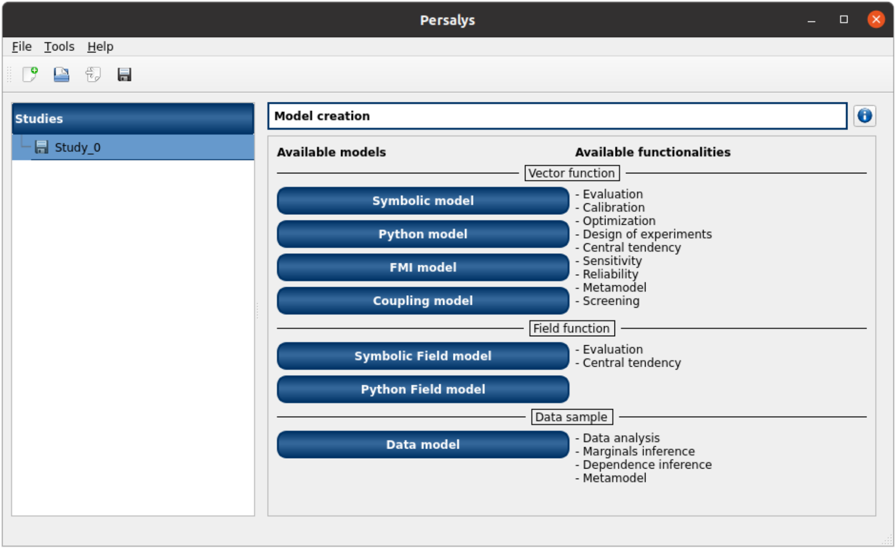
Persalys installation

- 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

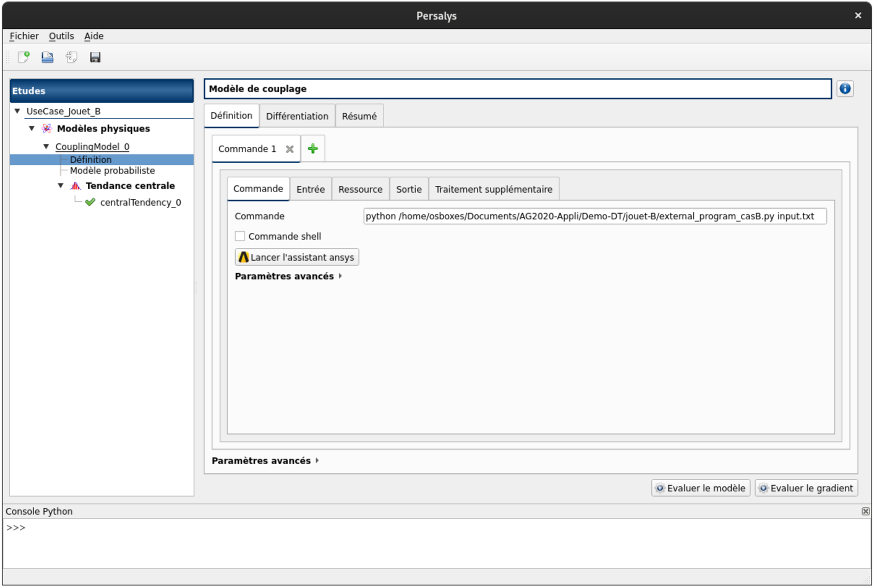
Models are viewed as a "black box" with inputs, outputs and a transfer function (TF). Persalys supports two model categories:

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

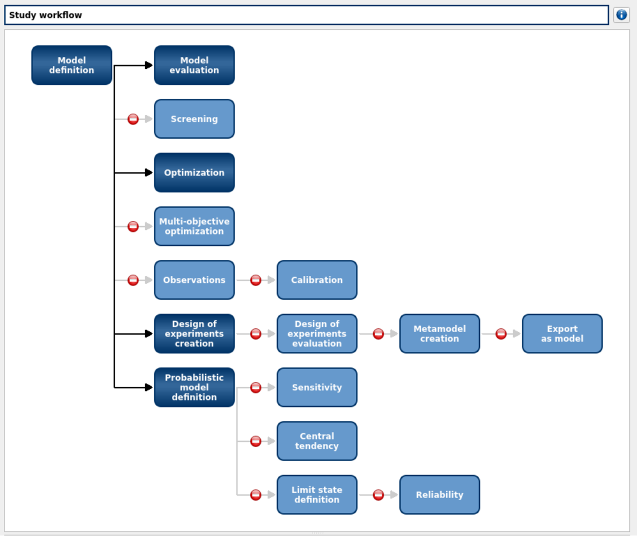
Vector to vector models can be of the following type:

- 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



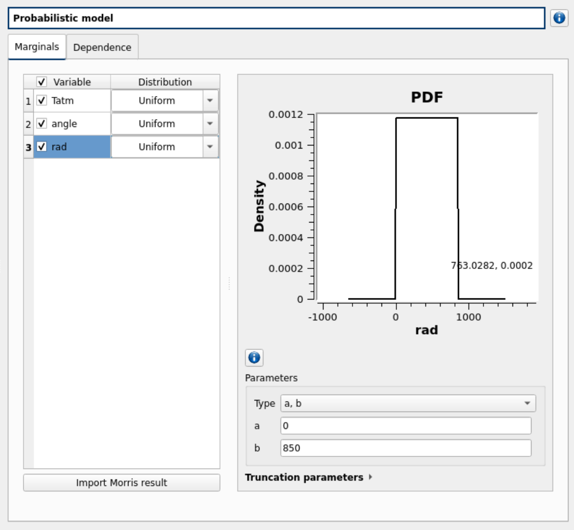
Study workflow



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

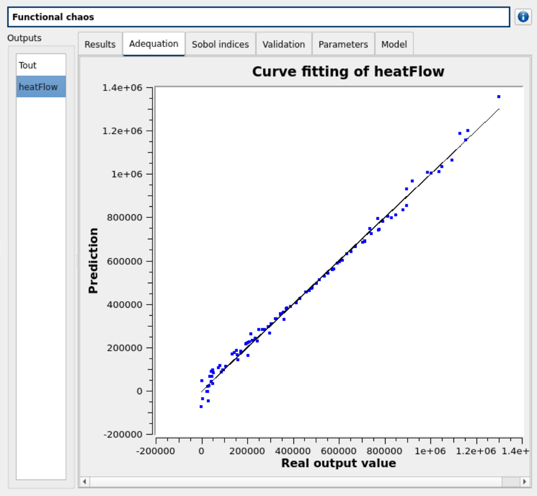
Probabilistic models

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`