… solutions for robust engineering

OpenTURNS

G. Blondet, Phimeca Engineering SA

‘HPC and Uncertainty Treatment – Examples with Open TURNS and Uranie’

EDF – Phimeca – Airbus Group – IMACS – CEA

PRACE Advanced Training Center – May, 10-12, 2021

Outline

- Overview of OpenTURNS (OT)
- Doc and Users
- OT in pictures
- OT in practice

Outline

What is Open TURNS?

- Partnership since 2005 between:

  EDF - R&D **EADS - Innovation Works** **Phimeca** **IMACS** (since 2014)

- **Open**-source initiative to **T**reat **U**ncertainties, **R**isks'**N** **S**tatistics
  - Open-source platform for uncertainty treatment
  - **Uncertainty propagation**
  - **Uncertainty quantification** and Uncertainty ranking
  - **Meta-model** building
  - working on Unix/Linux platform and Windows (since 2010)

- Open TURNS includes:
  - C++ scientific library including the methods for performing uncertainties treatment (statistic, reliability, etc.);
  - A python module : simplify your work with an interpreted language;
  - A complete documentation;
  - A website: http://openturns.github.io/

Uncertainty methodology (1/2)

Global Methodology of Treatment of Uncertainties

- developed first at EDF R&D in 1990 and then improved by contributions from other companies

Step A : Study Specification

➢ Uncertainty sources, model, variable of interest and criteria

Step B : Uncertainty Quantification

➢ Joint probability density function of the input uncertain parameters modeling

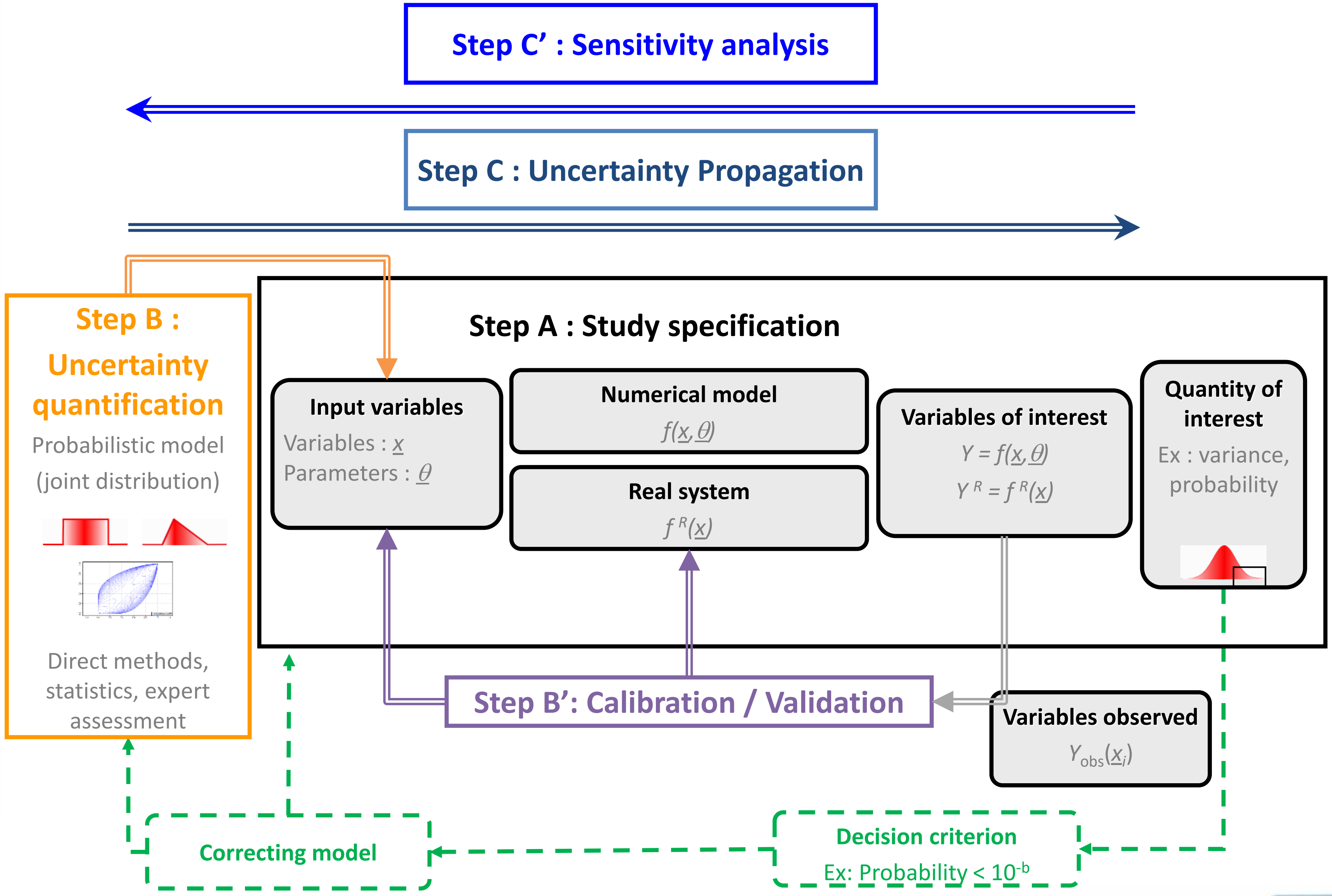
Step C : Uncertainty Propagation

➢ Variable of interest uncertainty assessment

Step C' : Uncertainty Ranking / sensitivity analysis

➢ Uncertainty sources ranking with respect to their influence on the variable of interest uncertainty

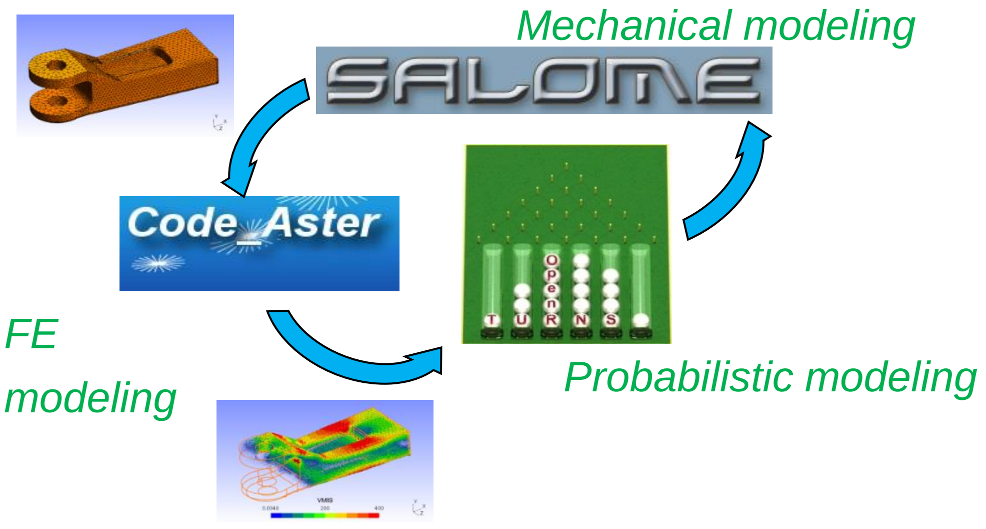
Uncertainty methodology (2/2)


Step C' : Sensitivity analysis
Step C : Uncertainty Propagation
Step B : Uncertainty quantification
Probabilistic model (joint distribution)
Direct methods, statistics, expert assessment
Step A : Study specification
Input variables
Variables : $\underline{x}$
Parameters : $\underline{\theta}$
Numerical model
$f(\underline{x},\underline{\theta})$
Real system
$f^R(\underline{x})$
Variables of interest
$Y = f(\underline{x},\underline{\theta})$
$Y^R = f^R(\underline{x})$
Quantity of interest
Ex : variance, probability
Step B': Calibration / Validation
Variables observed
$Y_{obs}(\underline{x}_i)$
Correcting model
Decision criterion
Ex: Probability < 10^{-b}

OpenTURNS features

Code linked to OpenTURNS

- **Interface with python functions ➔** to perform complex wrappers without compilation + parallelization functionalities
- Standard Interface for the wrappers of any complexity (distributed wrapper, binary data) development requiring the development of an external wrapper
- SalomeMeca compatible ➔ software including the 3 components to perform a mechanical and probabilistic data models coupling (linked to YACS)
- GUI of OpenTURNS within SalomeMeca



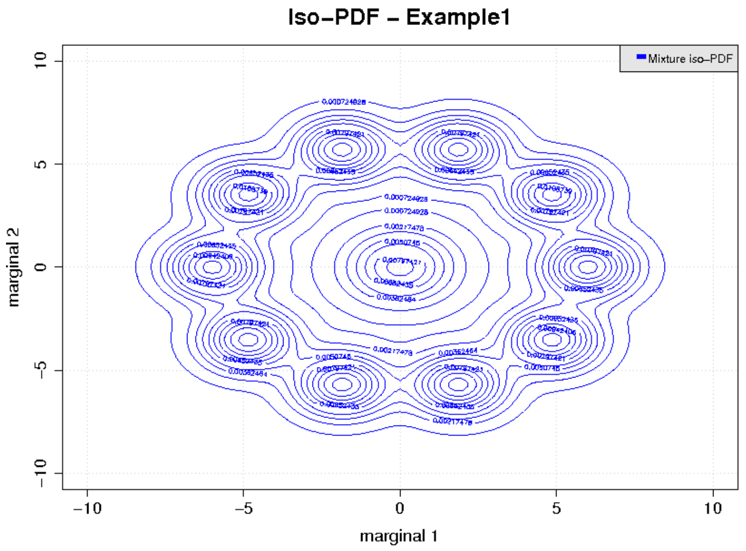
Uncertainty quantification with OpenTURNS

Estimation from data :

- Distribution fittings (parametric or not)
- Validation Tests (quantitative or graphical)
- Estimation of the dependence : copula, correlation coefficient
- Regression

Analytical modeling of joint distributions of dimension n :

- Combination Marginals + Copula
- Parametric distributions of dimension n
- Truncated distributions
- Stochastic process
- Non parametric distribution of dimension n: kernel fitting (n), Sklar Copula
- Linear combination of PDF
- Linear combination of random variables
- Random sum of independent discrete variables according to a Poisson process
- Etc.

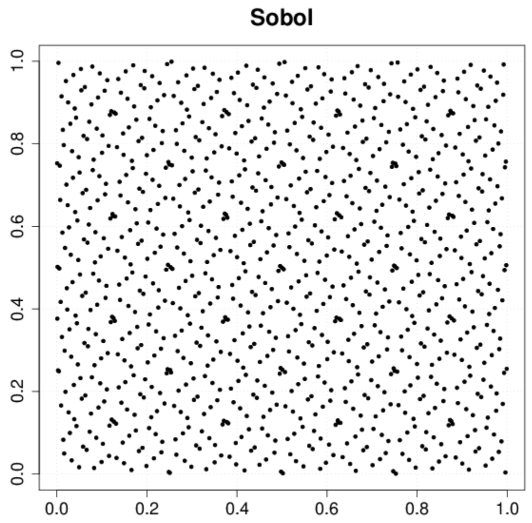


PHIMECA

Uncertainty propagation with OpenTURNS

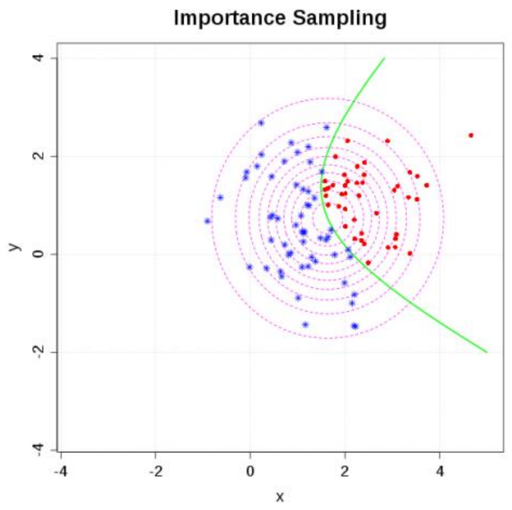
Sampling data:

- Random generator
- Stratified design of experiment
- Latin Hypercube Sampling
- Low Discrepancy Sequence
- Markov chain



Probability estimation:

- Isoprobabilistic transformation
- FORM / SORM
- Monte Carlo simulation
- Importance simulation
- Directional simulation
- Latin hypercube simulation
- Simulation algorithms

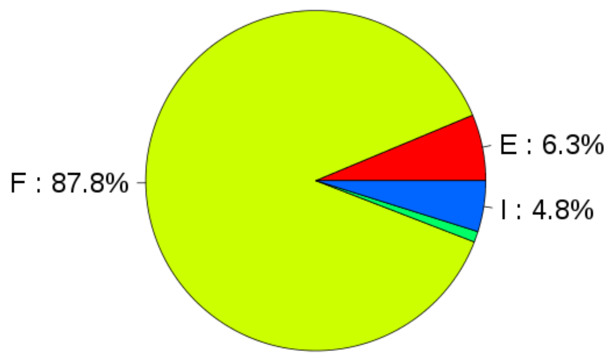


Uncertainty ranking with OpenTURNS

Ranking and sensitivity analysis:

- Importance factor from Taylor decomposition
- Ranking from correlation
- Sensitivity analysis
- Importance factor from reliability methods

Importance Factors from Design Point - Unnamed



Tools

- Optimization algorithm
- Response surface:
  - Parametric approximation
  - Functional chaos expansion
  - Kriging
- Graph

and Modules…

Innovative and recently implemented algorithms

- the most recent and efficient algorithms of non uniform distribution generation
  - Ziggurat method (2005) for the normal distribution
  - sequential reject algorithm (1993) for the binomial distribution,
  - Tsang & Marsaglia method (2000) for the gamma distribution,
  - Lebrun algorithm (2012) for the MultiNomial distribution,
- the most recent algorithms for evaluating the CDF
  - Marsaglia algorithm for the exact statistics of Kolmogorov (2003),
  - Benton et Krishnamoorthy algorithm for the distributions non centered Student and non centered Chi2 (2003).
- PhD results
  - Sparse chaos expansion polynomials : G. Blatman (EDF/R&D/MMC) (2010)
  - Accelerated simulation algorithm for the evaluation of low probabilities : M. Munoz (EDF/R&D/MRI) : (current dev)
  - Copulas for order statistics distributions: R. Lebrun (EADS) , Richard Fischer (EDF) (2013)

Outline

OpenTURNS: Doc and Users

- Several guides intended for users
  - **Installation** : Windows/Linux, from anaconda or sources
  - **API Reference** : Python docstring of most of the objects, arguments and methods in OpenTURNS and available in HTML.
  - **Examples Guide** : application of the whole Global Methodology on classical mechanical examples
  - **Reference Guide** : Theory of the methods implemented within OpenTURNS
  - **Contribute** : how to contribute to OpenTURNS, core code, modules …

- … and a sympathetic community :
  - Openturns.org : official web site
  - a particular page share to communicate about the software
  - the annual Users Day

OpenTURNS: Doc and Users

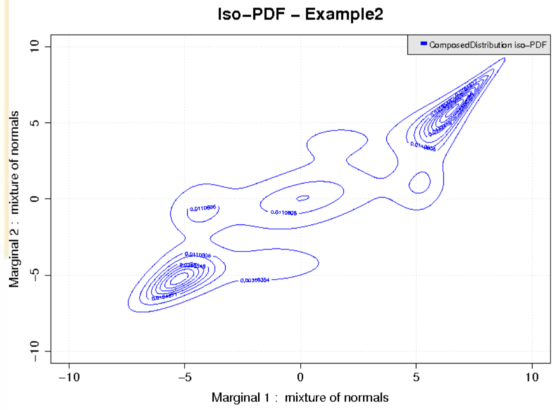
- Mailing list for users: users@openturns.org:
  - Ask any question relative to the installation and use of Open TURNS

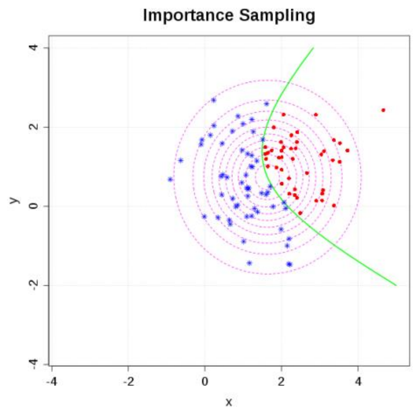
- Community tools:
  - https://openturns.discourse.group/
  - https://gitter.im/openturns/community

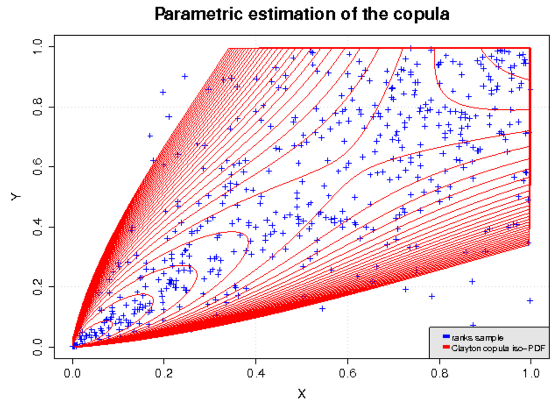
Outline

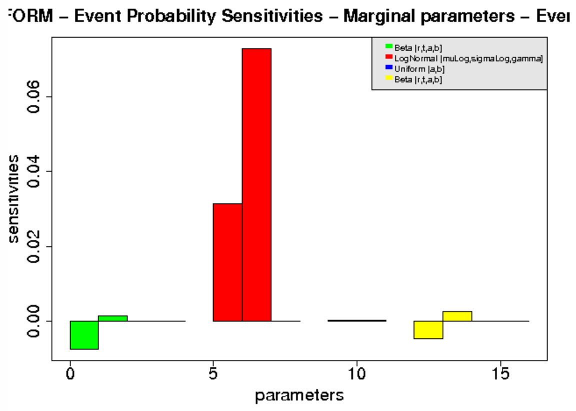
- Overview of OpenTURNS
- Doc and Users
- OT in pictures
- OT in practice

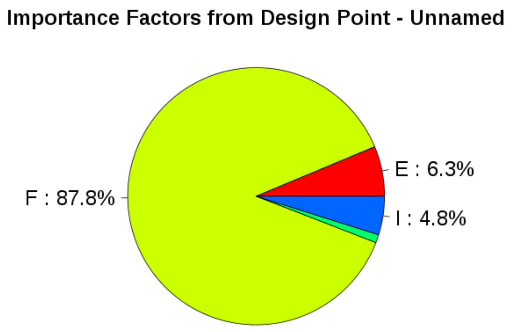
OpenTURNS in pictures

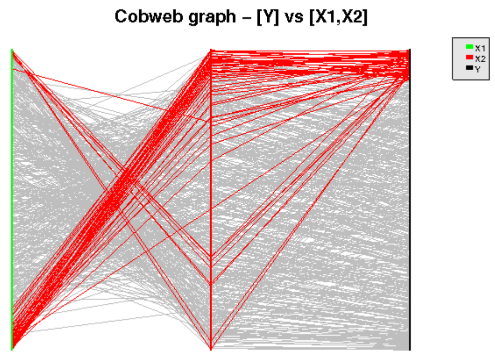




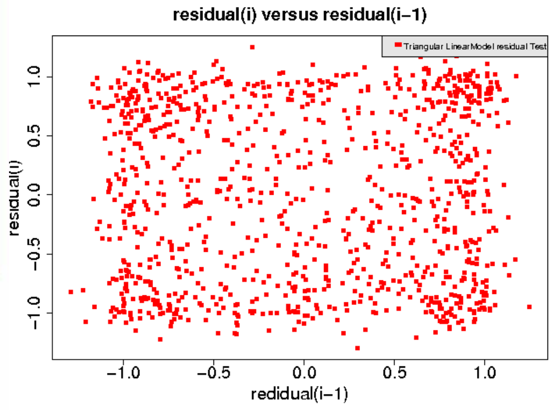


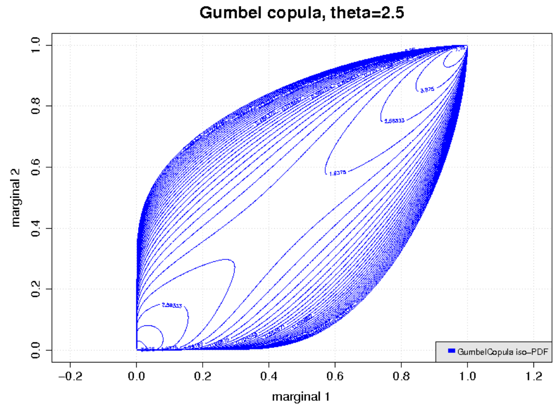


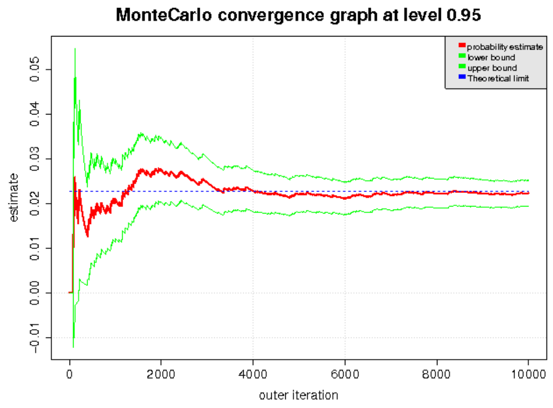


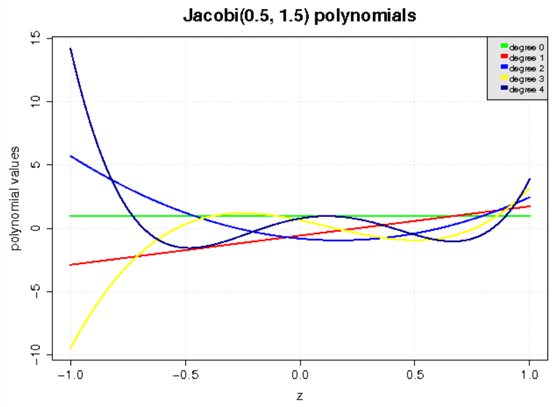


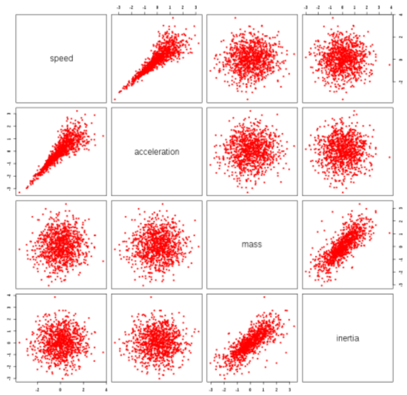
OpenTURNS in pictures

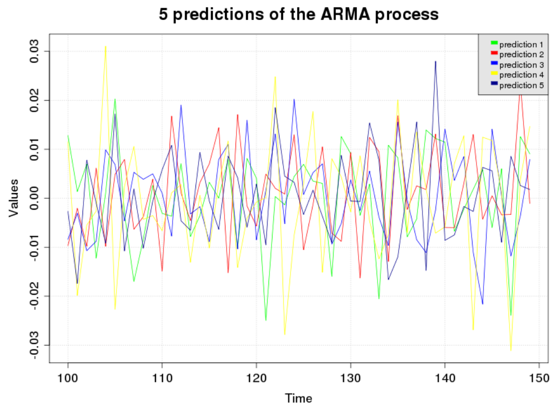




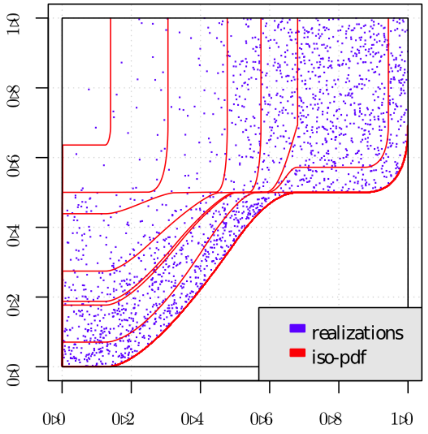


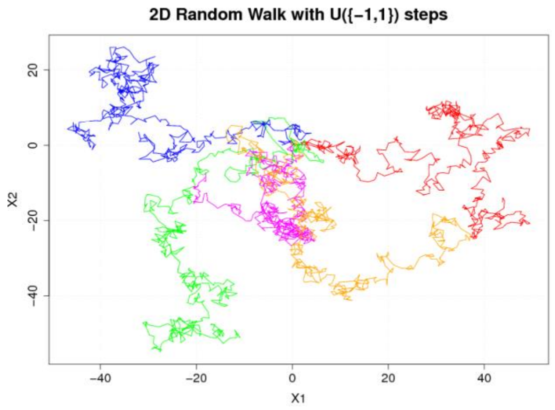


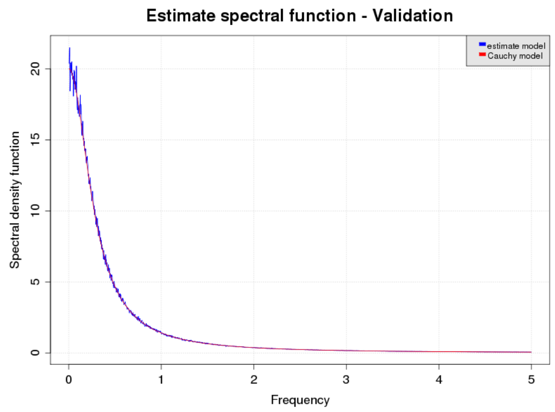


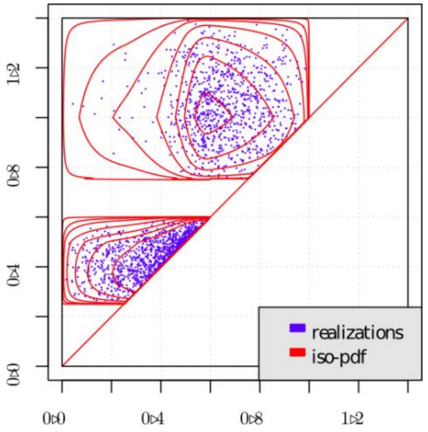


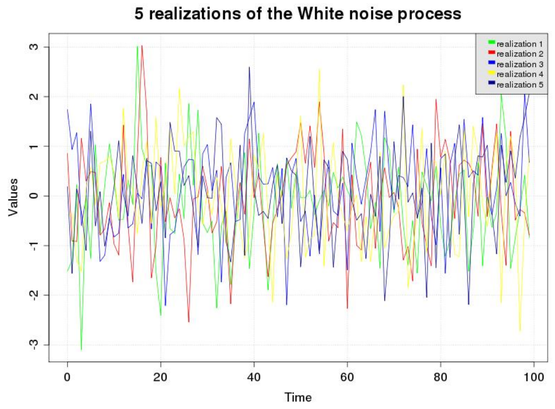
OpenTURNS in pictures

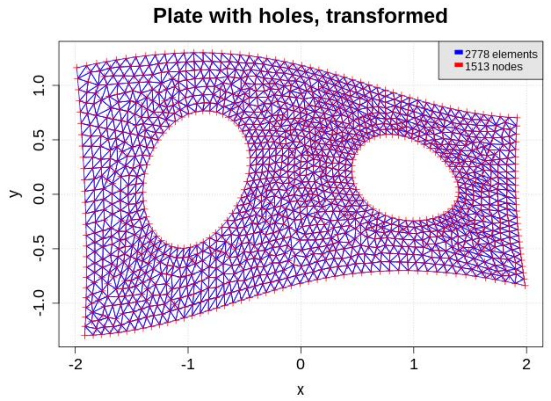












Outline

Basic commands in OT (1/3)

- Importation of OpenTURNS functionalities

```
import openturns as ot
```

- Mathematical objects

```
a = ot.Point(3)         # Vector of 3 components of dimension 1
S = ot.Sample(2, 3)     # Vector of 2 components of dimension 3
b = ot.Matrix(5 ,7)     # Matrix with 5 rows and 7 columns
d = ot.Tensor(3, 4, 5) # Tensor à 3 rows, 4 columns and 5 pages
d[2, 1, 3] = -2.0       # assign the value -2 to the 3rd row, 2nd column and 4th page, of d
```

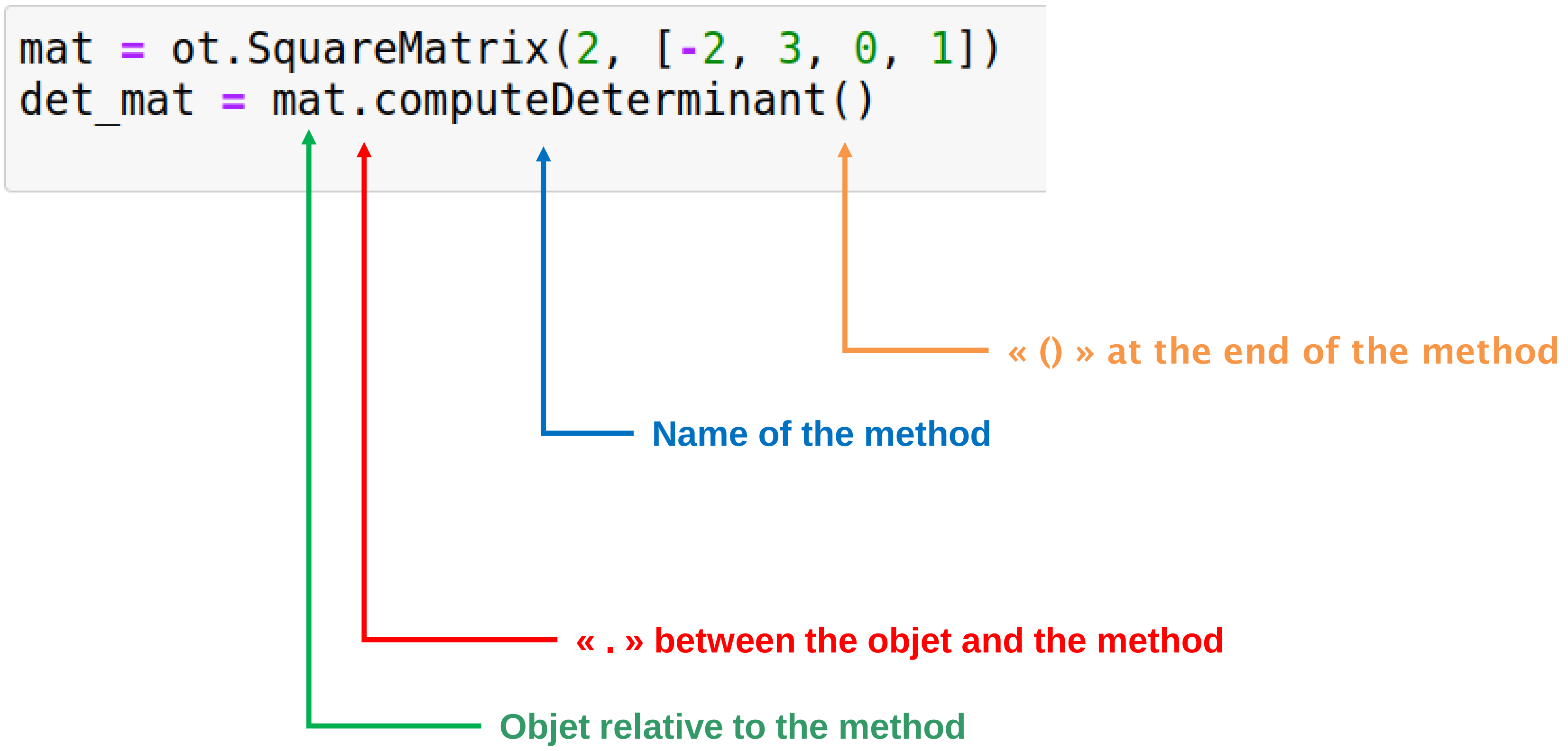
Basic commands in OT (2/3)

Methods

```
a = ot.Point(3)
a[0] = 2.
a[1] = -3.
a[2] = 5.
norm_a = a.norm() # Euclidean norm of the vector a

mat = ot.SquareMatrix(2) # Squared matrix of order 2
mat[0, 0] = -2.
mat[0, 1] = 3.
mat[1, 0] = 0.
mat[1, 1] = 1.
det_mat = mat.computeDeterminant() # Determinant of the matrix mat

y = ot.Point(2)
y[0] = 1.
y[1] = 5.
x = mat.solveLinearSystem(y) # Solve the system mat*x=y
```

Basic commands in OT (3/3)

- Methods

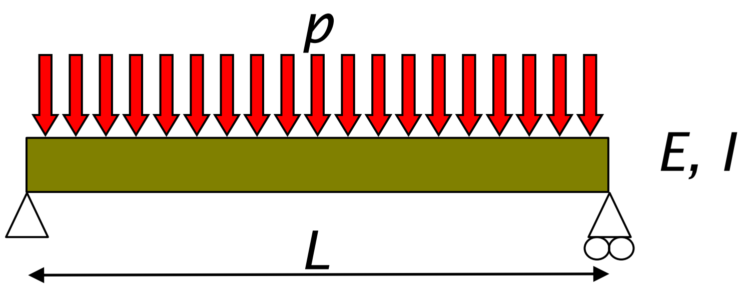


```
mat = ot.SquareMatrix(2, [-2, 3, 0, 1])
det_mat = mat.computeDeterminant()
```

« () » at the end of the method
Name of the method
« . » between the objet and the method
Objet relative to the method

Example of uncertainty propagatio

- Bending beam under uniform loading



- Maximal displacement

$$v_{\max} = \frac{5}{384}\frac{pL^4}{EI}$$

- Probabilistic model

parameter	Symbol	Distribution	Mean	Standard Deviation
Length [mm]	L	Lognormal	5000	50
Young modulus [MPa]	E	Lognormal	30000	4500
Inertia [mm^4]	I	Lognormal	10^9	10^8
Load [N/mm]	p	Lognormal	10	3

Model function

Defining a Function

```python
import openturns as ot
```

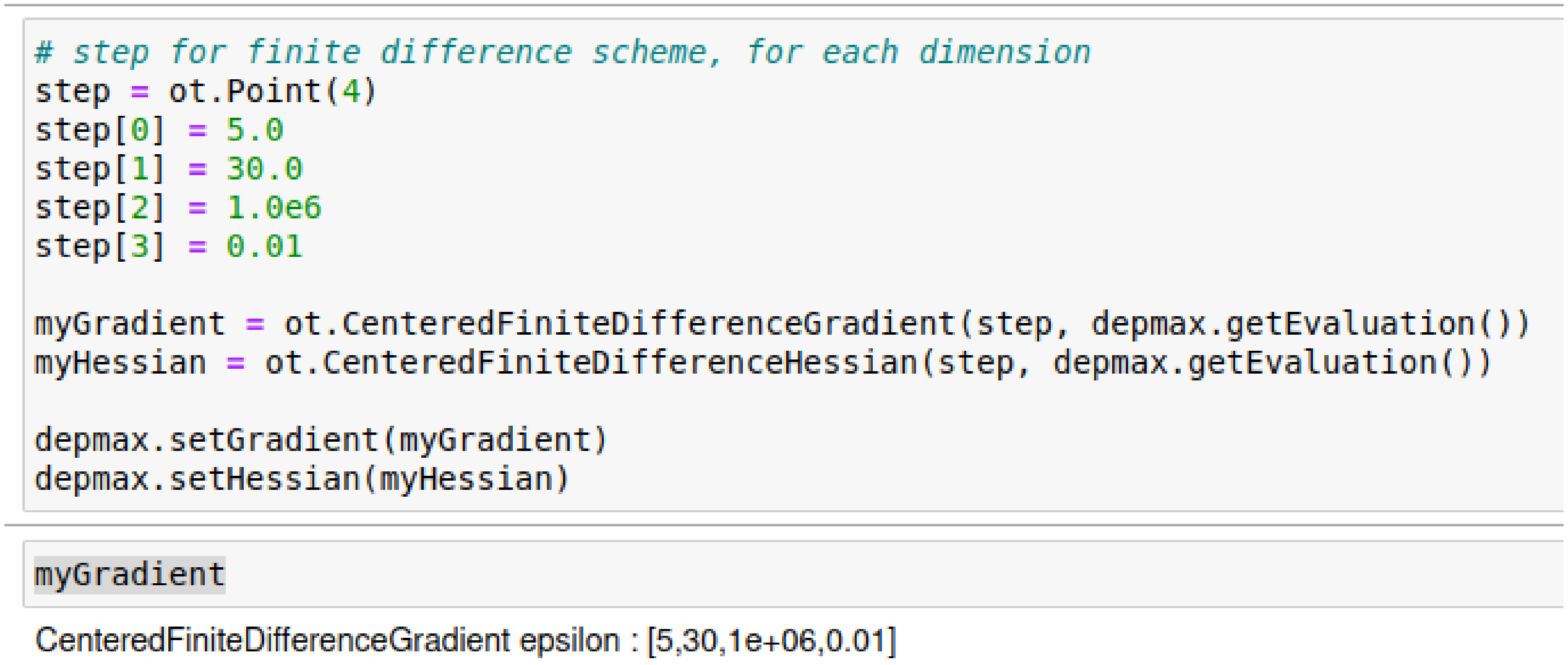
```python
# one way to do
class myfunction(ot.OpenTURNSPythonFunction):

    def __init__(self):
        ot.OpenTURNSPythonFunction.__init__(self, 4, 1)

    def _exec(self, X):
        '''
        Compute max displacement for the beam
        X -- array dim 1, 4 components
        '''
        dep_max = 5.0 / 384.0 * X[3] * X[0] ** 4 / (X[1] * X[2])
        return([dep_max])

depmax = ot.Function(myfunction())
print(depmax)
```

```
class=PythonEvaluation name=myfunction
```

```python
# another way
Depmax = ot.SymbolicFunction(["x1", "x2", "x3", "x4"],
                             ['5. / 384 * x4 * x1 / (x2 * x3)'])
```

Defining the derivatives

Centered Finite difference



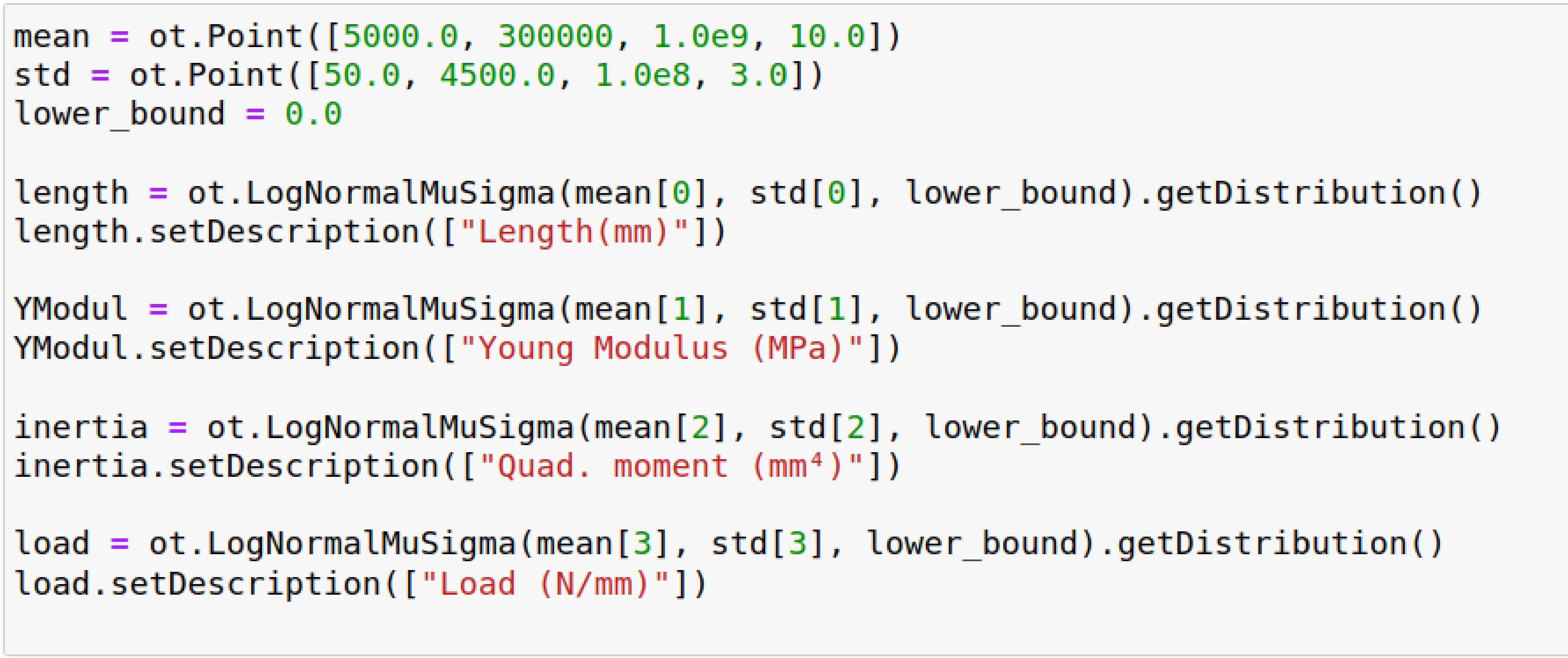
```
# step for finite difference scheme, for each dimension
step = ot.Point(4)
step[0] = 5.0
step[1] = 30.0
step[2] = 1.0e6
step[3] = 0.01

myGradient = ot.CenteredFiniteDifferenceGradient(step, depmax.getEvaluation())
myHessian = ot.CenteredFiniteDifferenceHessian(step, depmax.getEvaluation())

depmax.setGradient(myGradient)
depmax.setHessian(myHessian)
```

```
myGradient
```

```
CenteredFiniteDifferenceGradient epsilon : [5,30,1e+06,0.01]
```

Defining the probabilistic model

1 – Define the marginals



```python
mean = ot.Point([5000.0, 300000, 1.0e9, 10.0])
std = ot.Point([50.0, 4500.0, 1.0e8, 3.0])
lower_bound = 0.0

length = ot.LogNormalMuSigma(mean[0], std[0], lower_bound).getDistribution()
length.setDescription(["Length(mm)"])

YModul = ot.LogNormalMuSigma(mean[1], std[1], lower_bound).getDistribution()
YModul.setDescription(["Young Modulus (MPa)"])

inertia = ot.LogNormalMuSigma(mean[2], std[2], lower_bound).getDistribution()
inertia.setDescription(["Quad. moment (mm⁴)"])

load = ot.LogNormalMuSigma(mean[3], std[3], lower_bound).getDistribution()
load.setDescription(["Load (N/mm)"])
```

Defining the probabilistic model

2 – Define the composed distribution

```
Collection = ot.DistributionCollection(4)
Collection[0] = ot.Distribution(length)
Collection[1] = ot.Distribution(YModul)
Collection[2] = ot.Distribution(inertia)
Collection[3] = ot.Distribution(load)
```

```
Modelproba = ot.ComposedDistribution(Collection)
```

Defining the probabilistic model

2 bis – Define the correlation

If the variables are dependent, a copula can be added.

```
R = ot.CorrelationMatrix(3)
R[0, 1] = 0.25
R[1, 2] = 0.25
Copula = ot.NormalCopula(R)

Modelproba = ot.ComposedDistribution(Collection, Copula)
```

Propagation using MC simulations

3 - Get input and output

```
# generates a sample of 1000 points in the 4D space of input variables.
Input = Modelproba.getSample(1000)

# get outputs from input sample and the model.
DepMC = depmax(Input)
```

Result of the MC simulation

- Estimation of the 4 first moments of the displacement

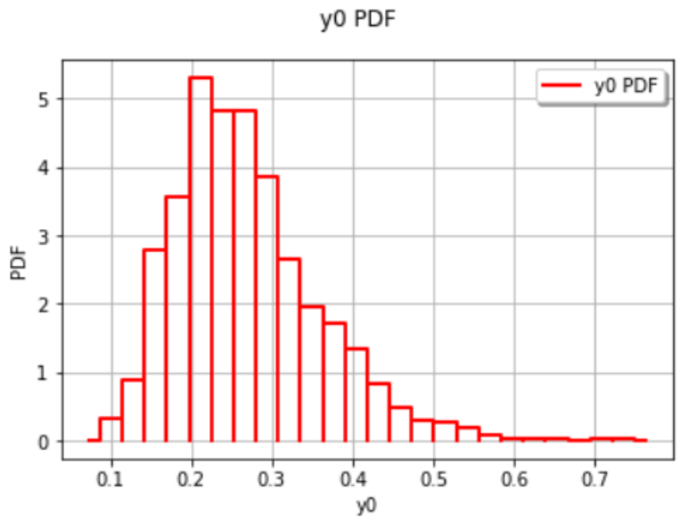
```
mean = DepMC.computeMean()
variance = DepMC.computeCovariance()
stdev= DepMC.computeStandardDeviation()
skewness = DepMC.computeSkewness()
kurtosis = DepMC.computeKurtosis()

print(mean, variance, stdev, skewness, kurtosis)
```

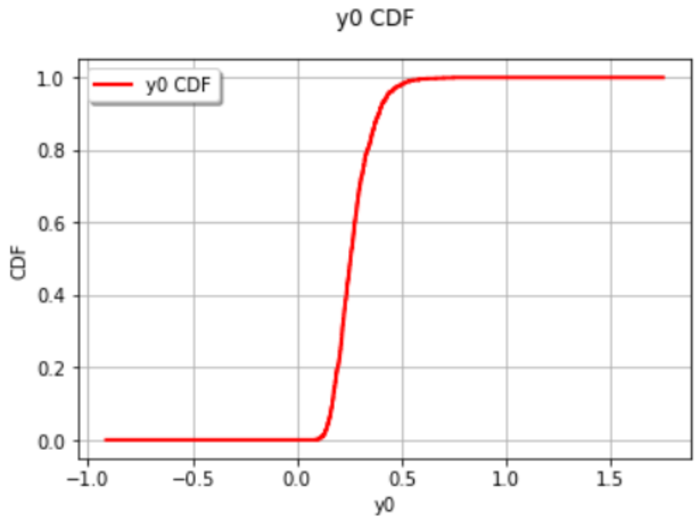
```
[0.268812] [[ 0.00804612 ]] [[ 0.0897002 ]] [1.03707] [4.89177]
```

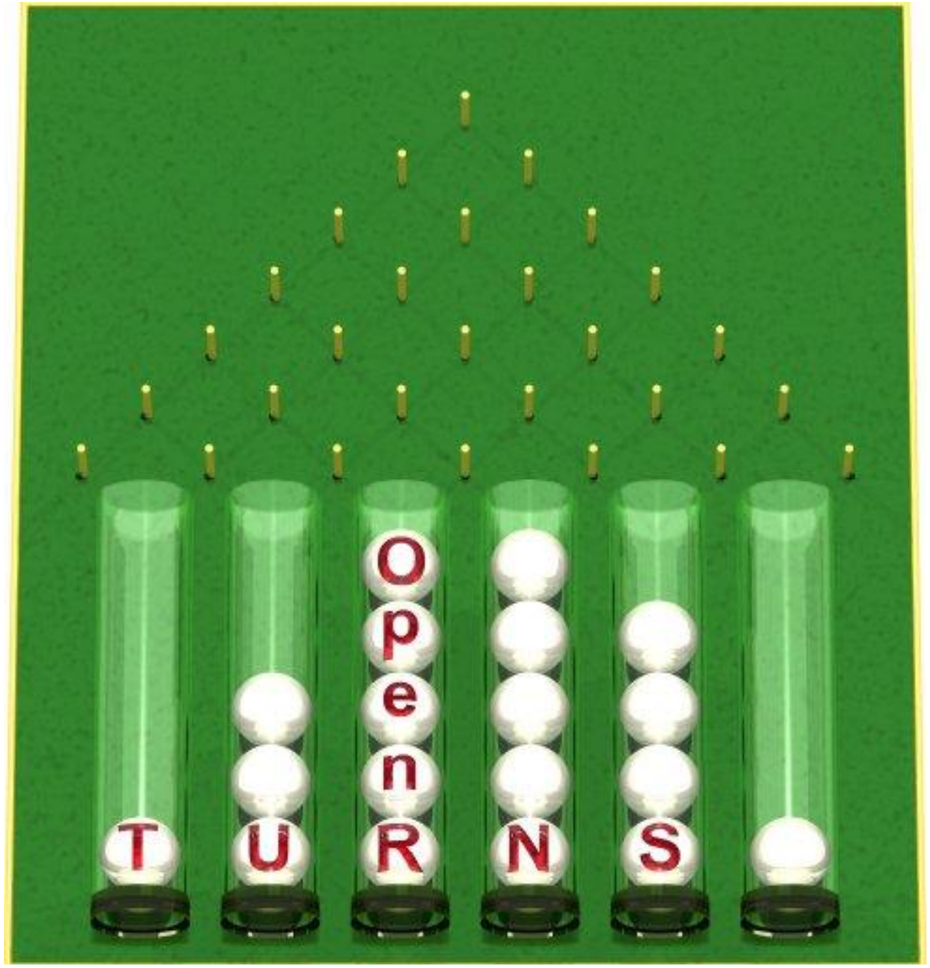
Histogram and empirical CDF

Graphs on the sample DepMC

```
import openturns.viewer as viewer
from matplotlib import pylab as plt

graph = ot.HistogramFactory().build(DepMC).drawPDF()
view = viewer.View(graph)
plt.show()

output_dist = ot.UserDefined(DepMC)
x_min = DepMC.getMin()[0] - 1.0
x_max = DepMC.getMax()[0] + 1.0
graph = output_dist.drawCDF(x_min, x_max)
view = viewer.View(graph)
plt.show()
```






O
p
e
n
T
U
R
N
S