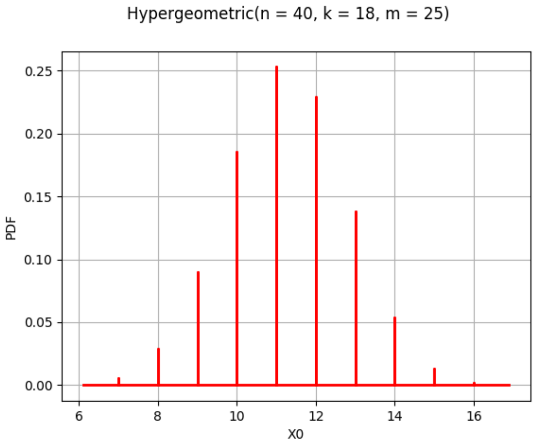
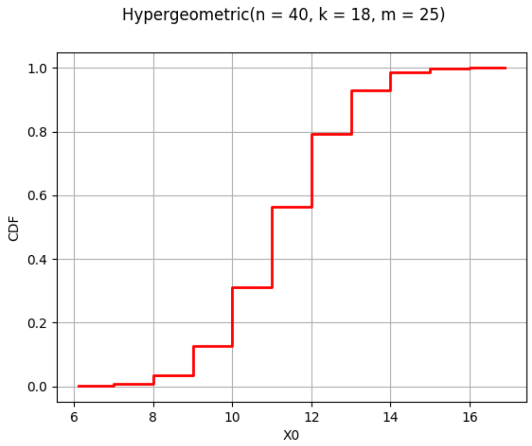
New features of the 1.12 & 1.13 releases

- Uncertainty modelling & quantification
  - ✓ Hypergeometric(n, k, m) where n is the population size, k the number of individuals with a given feature, m the size of the draw

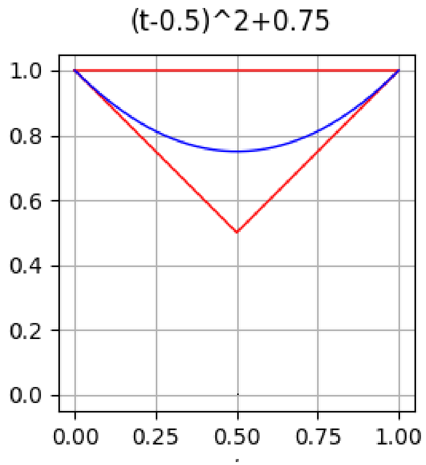


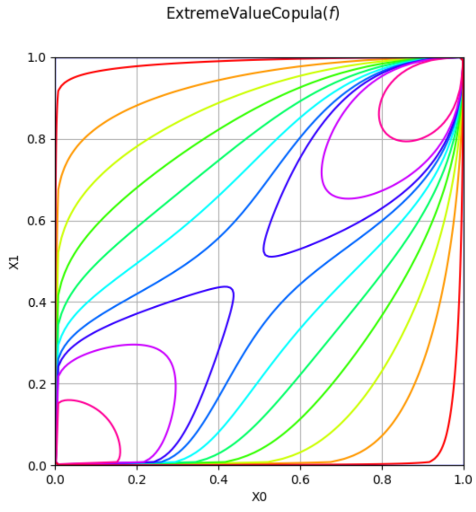


This distribution allows to model sampling without replacement.

- ✓ Extreme value copulas

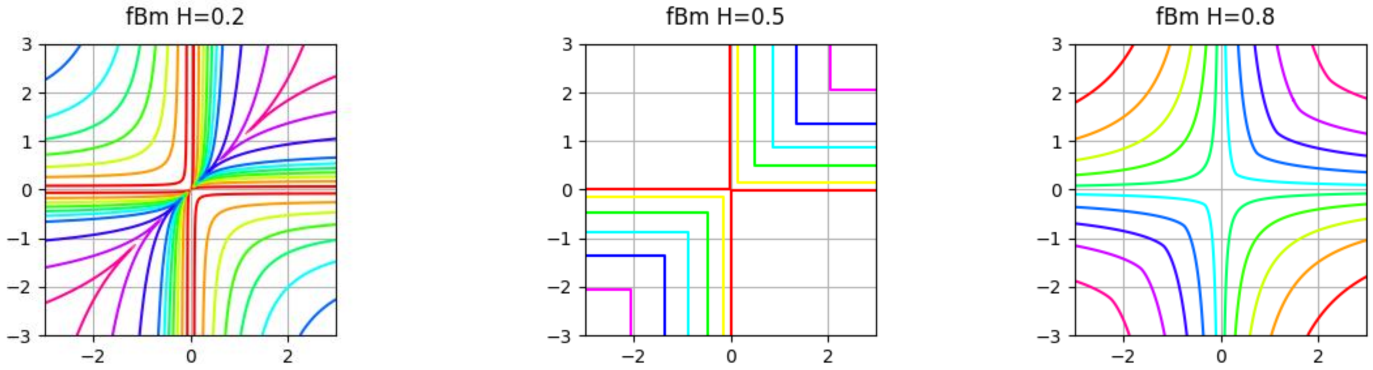
Useful to model joint extremes

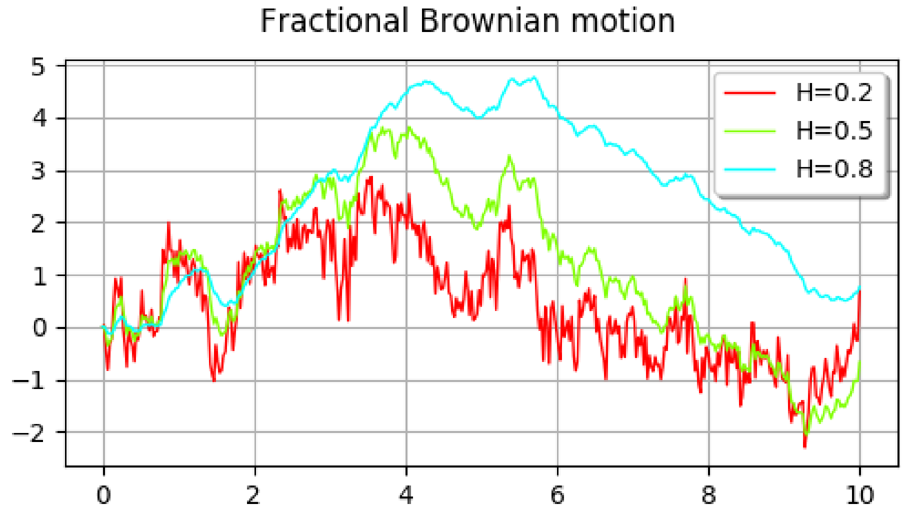




New features of the 1.12 & 1.13 releases

- Uncertainty modelling & quantification
  - ✓ FractionalBrownianMotionModel to sample Gaussian processes with irregular sample paths

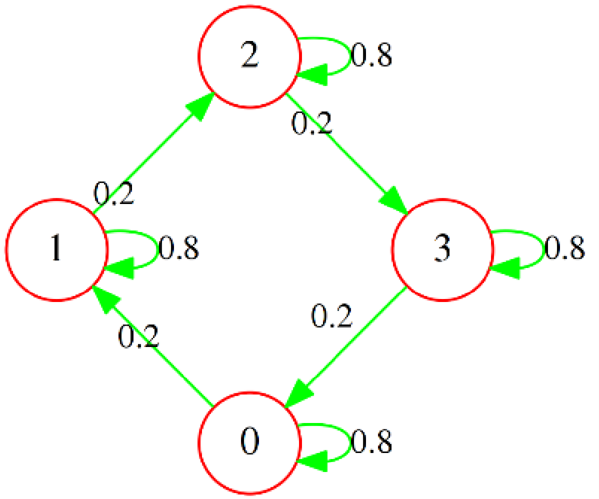


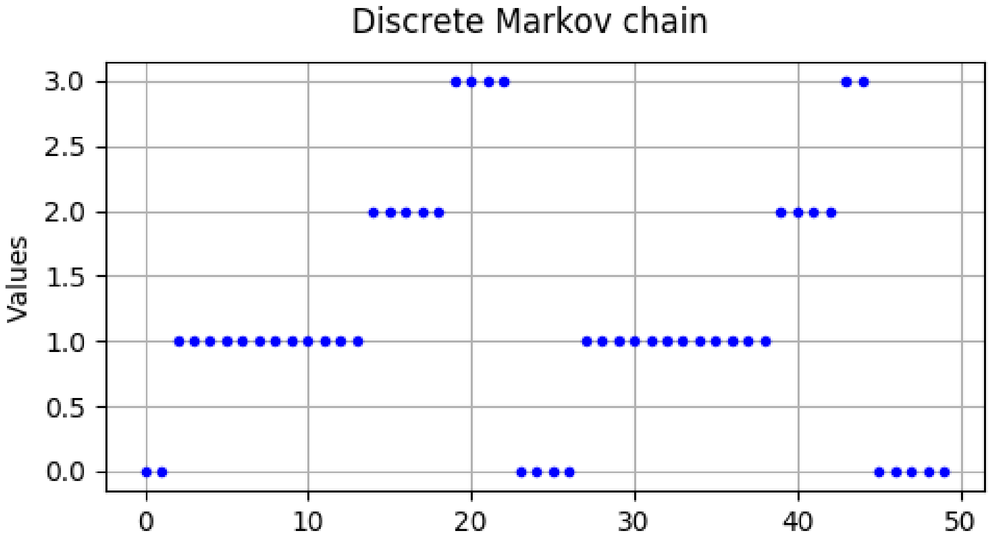


Useful eg. for the simulation of SDE

New features of the 1.12 & 1.13 releases

- Uncertainty modelling & quantification
  - ✓ DiscreteMarkovChain to model finite state Markov chains given a distribution for the initial state and a constant transition matrix.





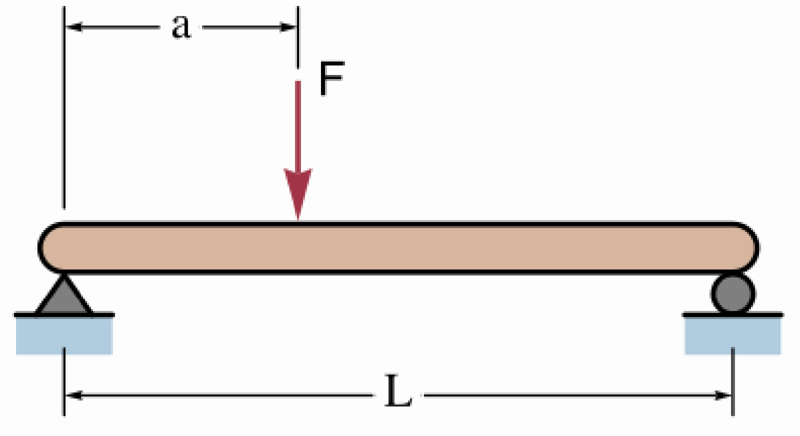
Countless uses in probabilistic modeling and stochastic algorithms

New features of the 1.12 & 1.13 releases

- Calibration
  - ✓ Given a parametric model and a set of noisy observations of its output, allows to compute a posterior distribution of the parameter and the error distribution in the following settings:

Parameter prior & dep	Dirac	Normal
Linear	LinearLeastSquaresCalibration	GaussianLinearCalibration
Nonlinear	NonLinearLeastSquaresCalibration	GaussianNonLinearCalibration

Example:
http://openturns.github.io/openturns/latest/examples/calibration/calibration_deflection_tube.html



calibrationFunction, observedInput, observedOutput, thetaPrior, parameterCovariance, errorCovariance)

New features of the 1.12 & 1.13 releases

- **Calibration**

Given:

✓the prior distribution $N(\theta_0, B)$ of $\theta = (a, D, d, L)$

✓the model between $y = f(x; \theta)$ with y=(right angle, deflection, left angle) and x=(F, E)

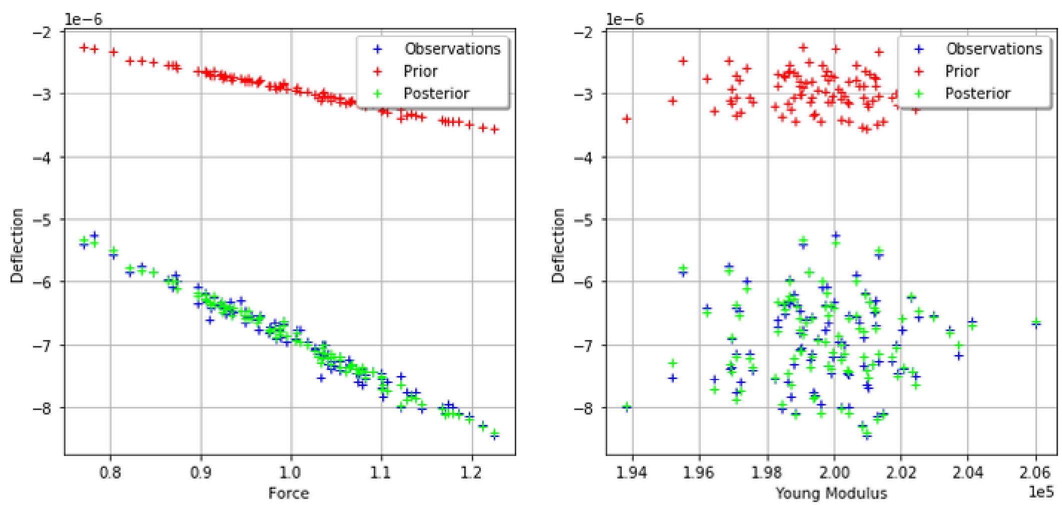
✓noisy observations y_i for given x_i and unknown θ

✓the distribution of the noise

Recover information about θ:

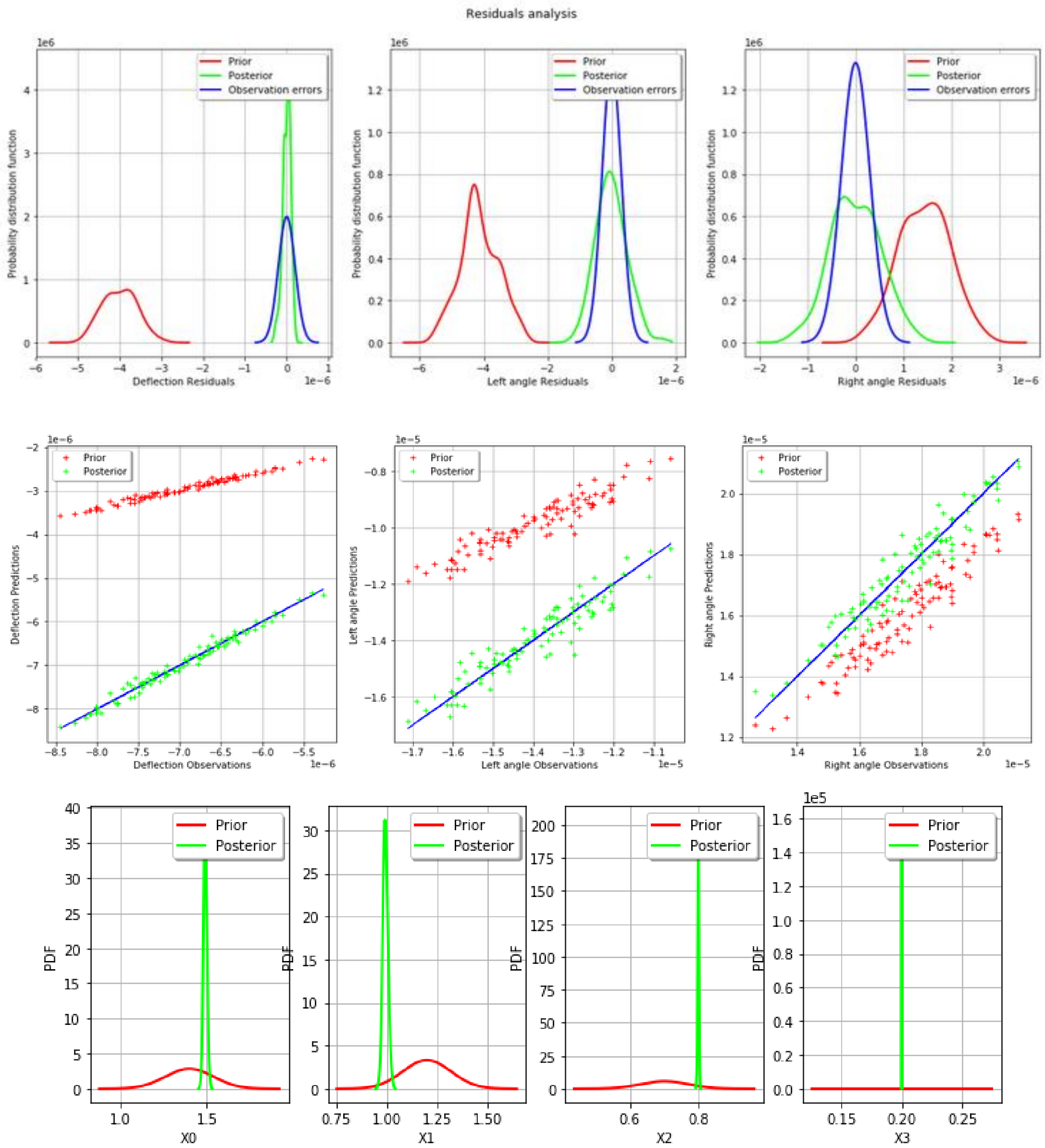
$\Theta^* = \mathrm{argmin}\ \|y - f(x; \theta)\|^2_R + \|\theta - \theta_0\|^2_B$ and $p(\theta|y) = K\exp(-[\|y - f(x; \theta)\|^2_R + \|\theta - \theta_0\|^2_B]/2)$

```
Algo = ot.GaussianNonLinearCalibration(calibrationFunction, observedInput,
       observedOutput, thetaPrior, parameterCovariance, errorCovariance)
Algo.run()
```



New features of the 1.12 & 1.13 releases

➢ Calibration



New features of the 1.12 & 1.13 releases

- Various improvements
  - ✓ Correct p-value for Kolmogorov-Smirnov tests with estimated parameters
  - ✓ Better statistical tests (access to the statistic, parameterized by the risk)
  - ✓ Extension of Sobol sequences to dimension 1111 for high dimension sampling
  - ✓ Huge improvement of Rosenblatt transformation performance (eg 4580 evals/s vs 37 evals/s for 5d mixtures)

New features of the 1.12 & 1.13 releases

- Sobol' and Expectation simulation algorithms
  - ✓ Iteratively sample and stop according to various critera (cov, ...)
  - ✓ Retrieve the estimate and its variance

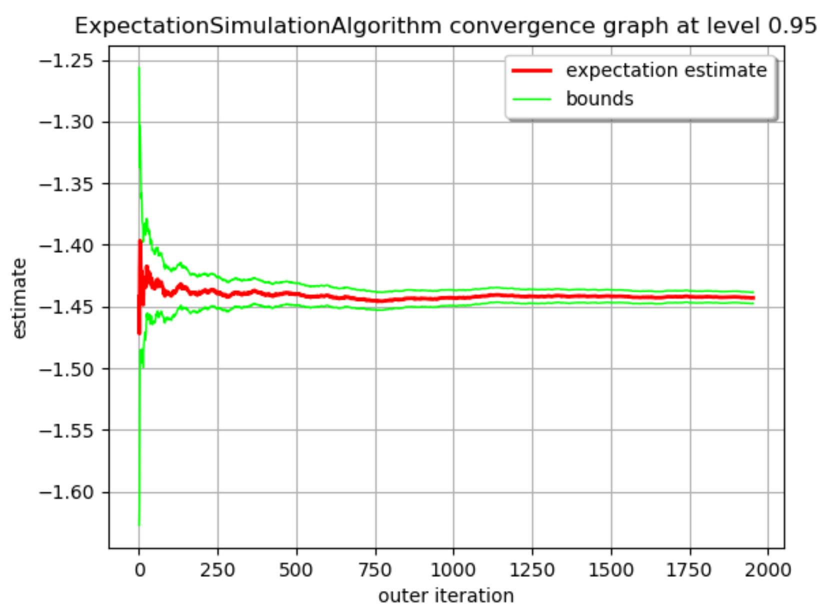
```
import openturns as ot

X = ot.RandomVector(model, distribution)

algo = ot.ExpectationSimulationAlgorithm(X)

algo.setMaximumOuterSampling(10000)
algo.setMaximumCoefficientOfVariation(0.05)
algo.setMaximumCoefficientOfVariationType('MAX')
algo.setMaximumStandardDeviation(0.001)
algo.setProgressCallback(progress)
algo.drawExpectationConvergence()
algo.run()

result = algo.getResult()
expectation = result.getExpectationEstimate()
expectation_dist = result.getExpectationDistribution()
```



New features of the 1.12 & 1.13 releases

- Optimization
  - ✓ OPT++ interior-point and Newton algorithms for general optimization problems
  - ✓ Nearest-point problem interface for FORM-like algorithms
  - ✓ Least squares problem interface used for calibration
  - ✓ Added CMinpack solver (Levenberg-Marquard, LS only problems)
  - ✓ Added Ceres solver (trust-region, line search methods, LS & general problems)

```
import openturns as ot


dim = 2
residualFunction = ot.SymbolicFunction(['x0', 'x1'], ['10*(x1-x0^2)', '1-
x0',...])
problem = ot.LeastSquaresProblem(residualFunction)
problem.setBounds(ot.Interval([-3.0] * dim, [5.0] * dim))

algo = ot.Ceres(problem, 'LEVENBERG_MARQUARDT')
algo.setStartingPoint([0.0] * dim)
algo.run()
result = algo.getResult()

x_star = result.getOptimalPoint()
y_star = result.getOptimalValue()
```

New features of the 1.12 & 1.13 releases

- Documentation updates
  - ✓ Completed legacy LaTeX doc migration with the stochastic process theoric section

We notice that for each fixed λ, the likelihood equation is proportional to the likelihood equation which estimates (β, σ^2). Thus, the maximum likelihood estimator for $(\beta(\lambda), \sigma^2(\lambda))$ for a given λ are:

$$\hat{\beta}(\lambda) = \frac{1}{N}\sum_{k=0}^{N-1} h_\lambda(x_k) \qquad (7)$$

$$\hat{\sigma}^2(\lambda) = \frac{1}{N}\sum_{k=0}^{N-1} (h_\lambda(x_k) - \beta(\lambda))^2$$

Substituting **(7)** into **(6)** and taking the $\log -$likelihood, we obtain:

$$\ell(\lambda) = \log L(\hat{\beta}(\lambda), \hat{\sigma}(\lambda), \lambda) = C - \frac{N}{2}\log\left[\hat{\sigma}^2(\lambda)\right] \; + \; (\lambda - 1)\sum_{k=0}^{N-1} \log(x_i)\,, \qquad (8)$$

where C is a constant.

The parameter $\hat{\lambda}$ is the one maximising $\ell(\lambda)$ defined in **(8)**.

API:

- See `BoxCoxTransform`
- See `InverseBoxCoxTransform`
- See `BoxCoxFactory`

Examples:

- See **Apply a Box-Cox transformation to a Field**

New features of the 1.12 & 1.13 releases

- Main API changes
  - ✓ The mesh becomes an attribute of Field functions which only exchange field values
  - ✓ New class ParametricPointToFieldFunction : parametric vector->Field function
  - ✓ Use of specialized RandomVector constructors (like the Function API change)
  - ✓ LinearModel/LinearModelFactory is deprecated (no more dependency to R)

```
import openturns as ot

>>> def pyF(X):
...     mesh = ot.RegularGrid(0.0, 0.1, 11)
...     size = 11
...     values = [ot.Point(X)*i for i in range(size)]
...     Y = ot.Field(mesh, values)
...     Y = [ot.Point(X)*i for i in range(size)]
...     return Y
>>> f = ot.PythonPointToFieldFunction(inputDim, mesh, outputDim, pyF)

>>> Y = ot.RandomVector(f, X)
>>> Y = ot.CompositeRandomVector(f, X)

>>> model = ot.LinearModelFactory(x, y).build()
>>> algo = ot.LinearModelAlgorithm(x, y); algo.run()
>>> model = algo.getResult().getMetaModel()
```

New features of the 1.12 & 1.13 releases

- New installation media
  - ✓ MacOS binaries available on Python package index (PyPI)
  - ✓ FreeBSD port published on freshports.org

```
pip install openturns
```

```
pkg install openturns
```