# Bayesian inference using MCMC in OpenTURNS

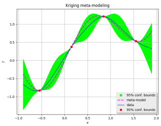
E. Songo [1] J. Muré [1] M. Keller [1]

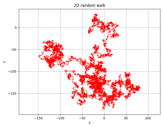
[1]EDF R&D. 6, quai Watier, 78401, Chatou Cedex - France, joseph.mure@edf.fr

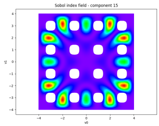
May 20th 2024, BEPU 2024, Lucca (Italy)

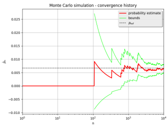
# OpenTURNS: www.openturns.org

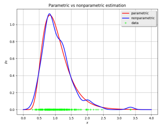
- Data analysis
  - Distribution fitting
  - Statistical tests
  - Estimate dependency and copulas
  - Estimate stochastic processes
- Probabilistic modeling
  - Dependence modeling
  - Univariate distributions
  - Multivariate distrbutions
  - Copulas
  - Processes
  - Covariance kernels
- Surrogate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



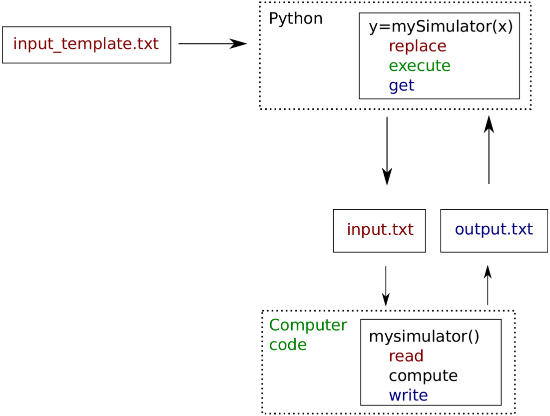



# Coupling OpenTURNS with computer codes

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

# OpenTURNS: Metropolis-Hastings

We want to sample from the distribution $\pi$ of a random variables $X$. Here is one step of the algorithm, starting from the point $x$:

1. Simulate a candidate $x' \sim q(\cdot|x)$ for some conditional distribution $q$.
2. Compute $\alpha(x'|x, y, z) = \min\left\{\frac{\pi(x')\, q(x|x')}{\pi(x) q(x'|x)}, 1\right\}$.
3. Simulate $u \sim \mathcal{U}(0, 1)$. If $u \leqslant \alpha(x'|x)$, then the next state is $x'$, otherwise it is $x$.

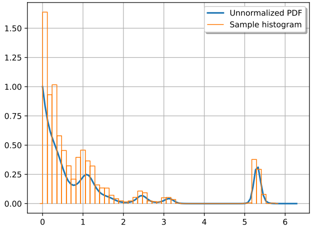
Throughout the presentation, our code is prefaced by:

```
import openturns as ot
import math as m
import numpy as np
```

# Random walk Metropolis Hastings

When $q(\cdot|x) = x + \mu$, where $\mu$ is a distribution that does not depend on $x$, the algorithm is called "Random walk Metropolis-Hastings" and $\mu$ is called the "proposal distribution".

## Sample from a nonstandard distribution[1]



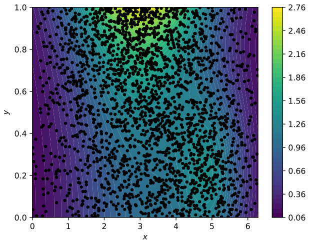

$$\pi(x)$$
$$\propto \frac{1}{2}\left(2+\sin(x)^2\right)$$
$$\exp\left[-\left(2+\cos(3x)^3+\sin(2x)^3\right)x\right]$$
$$\mathbf{1}_{[0,2\pi]}(x)$$

```
logdensity = ot.SymbolicFunction('x','log(2+sin(x)^2) - (2+cos(3*x)^3+sin(2*x)^3) * x')
support = ot.Interval([0.0], [2.0 * m.pi])
proposal = ot.Normal(0.0, 2.0) # mu
initialState = [3.0]
sampler = ot.RandomWalkMetropolisHastings(logdensity, support, initialState, proposal)
x = sampler.getSample(10000)
```

[1] Marin , J.M. and Robert, C.P. (2007). Bayesian Core: A Practical Approach to Computational Bayesian Statistics. *Springer-Verlag*, New York

# 2D Random walk Metropolis Hastings

## Sample from a 2D nonstandard distribution

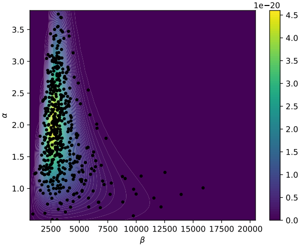



$$\pi(x) \propto \left( \exp\left[ -\frac{1}{4}(x-3)^2 + y^2 \right] + \exp\left[ -(x-5)^2 - 5\left(y - \frac{1}{5}\right)^2 \right] \right) \mathbf{1}_{[0,2\pi]}(x)\mathbf{1}_{[0,1]}(y)$$

```
logdensity = ot.SymbolicFunction(
    ["x", "y"], ["log(exp(-0.25 * (x-3)^2 + y^2) + exp(-(x-5)^2 - 5 * (y-0.2)^2))"]
)
support = ot.Interval([0.0, 0.0], [2.0 * m.pi, 1.0])
proposal = ot.Normal([0.0] * 2, [1.0, 0.2])
initialState = [3.0, 0.8]
sampler = ot.RandomWalkMetropolisHastings(logdensity, support, initialState, proposal)
x = sampler.getSample(50000)
```

# 2D Random walk Metropolis Hastings in a Bayesian setting

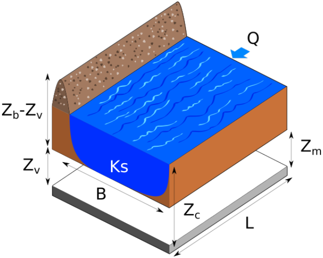
## Posterior distribution of the parameters of a Weibull model




$$\beta \sim \Gamma(k = 2, \lambda = 2 \cdot 10^{-4})$$
$$\alpha \sim \mathcal{U}(0.5, 3.8)$$
$$T|\beta, \alpha \sim \mathcal{W}(\beta, \alpha, 0)$$
$$F_{\mathcal{W}}(t) = 1 - \exp\left[-\left(\frac{t-0}{\beta}\right)^{\alpha}\right]$$

```
alpha_min, alpha_max, a_beta, b_beta = 0.5, 3.8, 2.0, 2.0e-4
priorMarginals = [ot.Gamma(a_beta, b_beta), ot.Uniform(alpha_min, alpha_max)]
prior = ot.ComposedDistribution(priorMarginals)
proposal = ot.Normal([0.0]*2, [0.1*m.sqrt(a_beta/b_beta**2), 0.1*(alpha_max-alpha_min)])
initialState = [a_beta / b_beta, 0.5 * (alpha_max - alpha_min)]
sampler = ot.RandomWalkMetropolisHastings(prior, initialState, proposal)

conditional = ot.WeibullMin()
Tobs = [[4380], [1791], [1611], [1291]]

# WeibullMin expects beta, alpha, and localization, but the prior is only on beta, alpha
linkFunction = ot.SymbolicFunction(["beta", "alpha"], ["beta", "alpha", "0"])
sampler.setLikelihood(conditional, Tobs, linkFunction)
sample = sampler.getSample(100000)
```

# A flood model



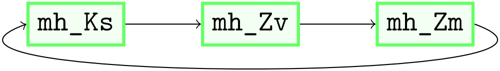

$$\forall 1 \leq i \leq 8, H^{(i)} \sim \mathcal{N}\left(G(Q^{(i)}, K_s, Z_v, Z_m), \frac{1}{2}\right)$$

$$K_s \sim \mathcal{N}(20, 5)$$
$$Z_v \sim \mathcal{N}(49, 1)$$
$$Z_m \sim \mathcal{N}(51, 1)$$

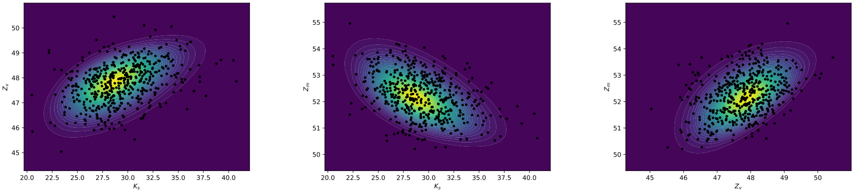
```
Qobs = [[2097], [1448], [1516], [2173], [387], [3016], [651], [541]]
Hobs = [[3.4], [2.5], [2.7], [3.5], [1.0], [4.2], [1.6], [1.6]]

def flooding(X):
    L = 5.0e3
    B = 300.0
    Q, K_s, Z_v, Z_m = X
    alpha = (Z_m - Z_v) / L
    if alpha < 0.0 or K_s <= 0.0:
        H = np.inf
    else:
        H = (Q / (K_s * B * np.sqrt(alpha))) ** (3.0 /
    5.0)
    return [H, 0.5]

functionG = ot.PythonFunction(4, 2, flooding)

# Q (input #0) is not calibrated
linkFunction = ot.ParametricFunction(functionG, [0], [100])
```

```
conditional = ot.Normal()

parameterPriorMean = [20.0, 49.0, 51.0]
parameterPriorSigma = [5.0, 1.0, 1.0]
prior = ot.Normal(parameterPriorMean, parameterPriorSigma)

initialState = parameterPriorMean
```

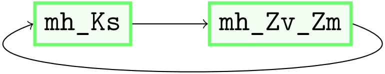
# Single component Random Walk Metropolis-Hastings



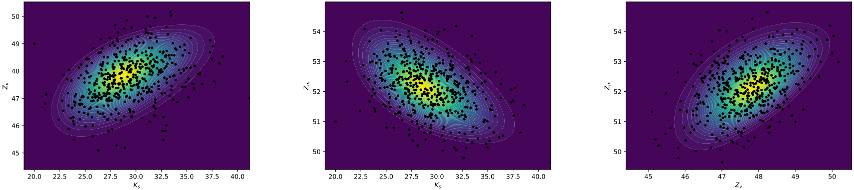

```python
mh_coll = [
  ot.RandomWalkMetropolisHastings(prior, initialState, ot.Uniform(-5.0, 5.0), [0]),
  ot.RandomWalkMetropolisHastings(prior, initialState, ot.Uniform(-1.0, 1.0), [1]),
  ot.RandomWalkMetropolisHastings(prior, initialState, ot.Uniform(-1.0, 1.0), [2]),
]

for mh in mh_coll:
    mh.setLikelihood(conditional, Hobs, linkFunction, Qobs)

sampler = ot.Gibbs(mh_coll) # NB: the order can be made random: cf. setUpdatingMethod
sample = sampler.getSample(10000)
```



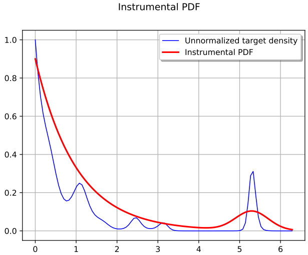
# Blocks of components can be considered



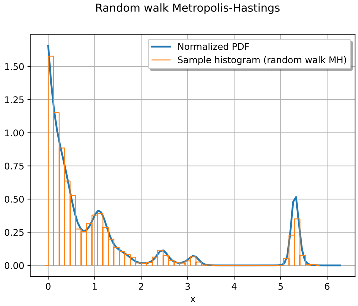

```
mh_coll = [
  ot.RandomWalkMetropolisHastings(prior, initialState, ot.Uniform(-5.0, 5.0), [0]),
  ot.RandomWalkMetropolisHastings(prior,
                                  initialState,
                                  ot.ComposedDistribution([ot.Uniform(-1.0,1.0)]*2),
                                  [1, 2])
]
for mh in mh_coll:
    mh.setLikelihood(conditional, Hobs, linkFunction, Qobs)
sampler = ot.Gibbs(mh_coll) # NB: the order can be made random: cf. setUpdatingMethod
sample = sampler.getSample(10000)
```

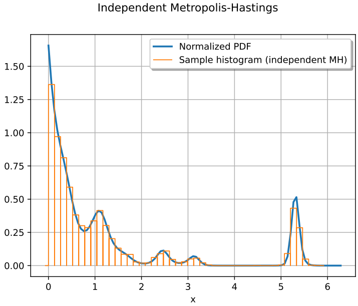


# Independent Metropolis-Hastings: $q(\cdot|x) = \mu$



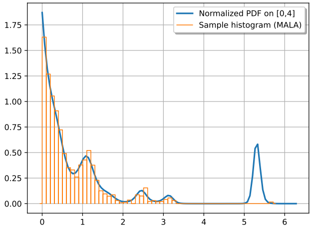

```
logdensity = ot.SymbolicFunction('x','...') # replace ...
support = ot.Interval([0.0], [2.0 * m.pi])
initialState = [3.0] # unimportant for independent MH

exp = ot.Exponential(1.0)
unif = ot.Normal(5.3, 0.4)
instrumental = ot.Mixture([exp, unif], [0.9, 0.1])

independentMH = ot.IndependentMetropolisHastings(
  logdensity, support, initialState, instrumental
)
x = independentMH.getSample(10000)
```



# User-defined Metropolis-Hastings: $q(\cdot|x) = \mu(x)$

## Metropolis adjusted Langevin algorithm[2] implementation




With $h > 0$ a fixed parameter:

$$q(\cdot|x) = \mathcal{N}\left(x + \frac{h}{2}\frac{d}{dx}[\log(\pi(x)], \sqrt{h}\right)$$

```
from openturns.experimental import UserDefinedMetropolisHastings
logdensity = ot.SymbolicFunction('x','log(2+sin(x)^2) - (2+cos(3*x)^3+sin(2*x)^3) * x')
support, proposal, initialState = ot.Interval([0.0], [2.0 * m.pi]), ot.Normal(), [2.5]
h = 0.5
std_deviation = m.sqrt(h)

def python_link(x):
    derivative_log_density = logdensity.getGradient().gradient(x)[0, 0]
    mean = x[0] + h / 2 * derivative_log_density
    return [mean, std_deviation]
link = ot.PythonFunction(1, 2, python_link)

mala = UserDefinedMetropolisHastings(logdensity, support, initialState, proposal, link)
z = mala.getSample(10000)
```

[2] Robert, C. P. *The Metropolis-Hastings algorithm*. arXiv preprint arXiv:1504.01896, 2015

# Application: airflow rate in a depressurized room

$$g(\xi, \theta_0, \theta_1) = 0.6 \times 3600 \times \theta_0 \left(\frac{2}{1.8}\xi\right)^{\theta_1}$$

For $1 \leqslant i \leqslant 233$:

$$\begin{cases} X_i &= \quad \xi_i + Z_i \\ Y_i &= g(\xi_i, \theta_0, \theta_1) + E_i \end{cases}$$

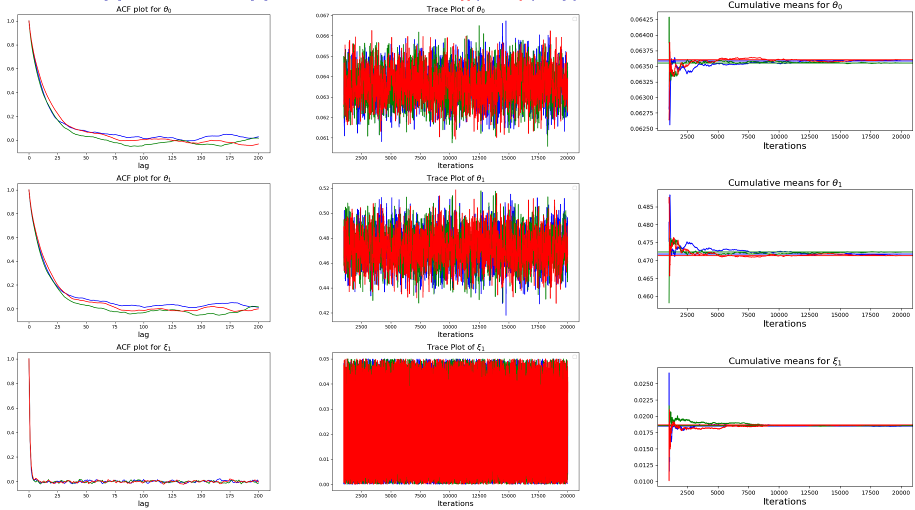
- Input: $X_i$ (pressure difference – *bars*)
- Output: $Y_i$ (airflow rate – $m^3/h$)
- Parameter: $\theta_0$ (area – $m^2$)
- Parameter: $\theta_1$ (exponent)

- $Z_i \sim U(-0.05, 0.05)$
- $E_i \sim N(0, \sigma_E^2)$, $\sigma_E^2 \sim 1/\sigma_E^2$
- $\theta_0 \sim U(0, 2)$
- $\theta_1 \sim U(0, 2)$

Strategy: $\sigma_E^2$ averaged out analytically, the rest sampled using Gibbs with:

- Random walk Metropolis-Hastings on $\theta_0$, step is tuned during burn-in.
- Random walk Metropolis-Hastings on $\theta_1$, step is tuned during burn-in.
- Independent MH on each $Z_i$ with the prior as proposal.

# Convergence diagnostics for $\theta_0$, $\theta_1$, $\xi_1$ with 3 chains




(a) ACF plot

(b) Trace plot

(c) Convergence plot of ergodic means

# Posterior distribution

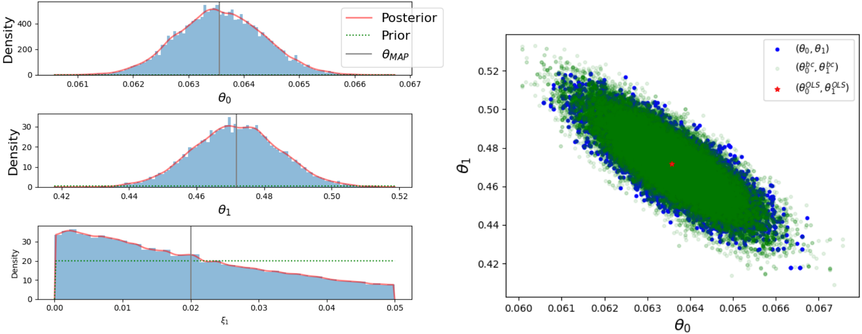



Figure: Left: Prior and posterior distributions of $\theta_0$, $\theta_1$ and $\xi_1$. Right: Scatter plot of the sample of $(\theta_0, \theta_1)$ from the joint posterior distribution (solid blue dots), from a simplified posterior where all $Z_i = 0$ (transparent green dots), alongside the Ordinary Least Squared estimator (red star)

# Conclusion

OpenTURNS provides an MCMC sampling framework through the following classes:

- MetropolisHastings variants:
  - RandomWalkMetropolisHastings
  - IndependentMetropolisHastings
  - UserDefinedMetropolisHastings
  - RandomVectorMetropolisHastings (not shown in this presentation)
- Gibbs

These classes can be freely combined to sample from nonstandard distributions in a “smart” manner.

In a Bayesian setting, this framework allows users to create and implement the MCMC algorithm most suited to a particular posterior distribution.