OpenTURNS release highlights

J. Pelamatti (EDF), J. Schueller (Phimeca)

UserDay #15, 10 June 2022, EDF Lab

Overview

New features since last year in releases:

- v1.18: fall 2021
- v1.19: spring 2022

Contents

HSIC sensitivity indices (finally!)

- *Given-data* sensitivity indices (applicable on a user-provided sample)
- Perform fairly well when dealing with small (and/or high dimensional) samples
- Provide HSIC and normalized R2-HSIC indices $\rightarrow$ useful for variable ranking
- Perform statistical tests $\rightarrow$ use p-values for screening of influential/non-influential variables
  - Asymptotic test
  - Permutation-based test

3 different types of analyses available

- Global Sensitivity Analysis (GSA) $\rightarrow$ *what are the influential variables on the output, globally?*
- Target Sensitivity Analysis (TSA) $\rightarrow$ *what are the influential variables when crossing a given threshold?*
- Conditional Sensitivity Analysis (CSA) $\rightarrow$ *what are the influential variables inside a given critical domain?*

A quick overview of the API

4 essential elements :

- An HSIC estimator
  - GSA/TSA/CSA
- 2 Data samples
  - p-dimensional input sample
  - 1-dimensional output sample
- A p+1-dimensional list of covariance models
- A type of HSIC statistic
  - U-statistics (unbiased)
  - V-statistics (biased, but asymptotically unbiased)

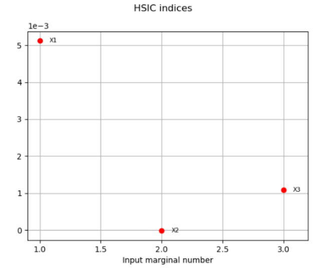
2 additional elements :

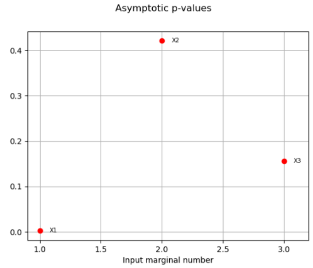
- TSA $\rightarrow$ Filter function
- CSA $\rightarrow$ Weight function

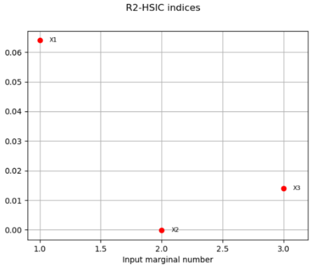
Example : Global sensitivity analysis on the Ishigami function

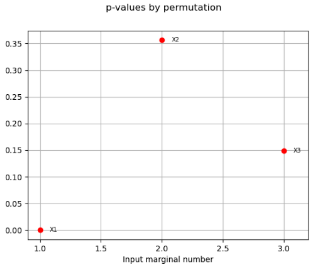
```
estimatorType = ot.HSICUStat()

GSAEstimator = ot.HSICEstimatorGlobalSensitivity(
    covarianceModelCollection, X, Y, estimatorType)

GSAEstimator.run()

graph1 = GSAEstimator.drawHSICIndices()
graph2 = GSAEstimator.drawPValuesAsymptotic()
graph3 = GSAEstimator.drawR2HSICIndices()
graph4 = GSAEstimator.drawPValuesPermutation()
```

Example : Global sensitivity analysis on the Ishigami function





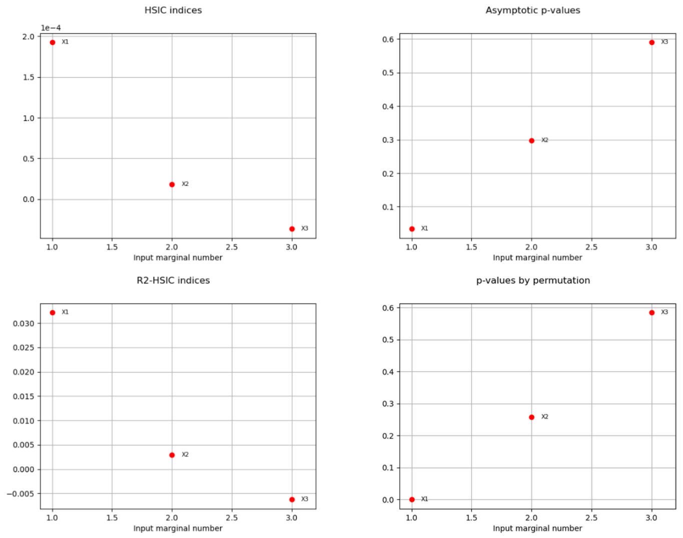




Example : Target sensitivity analysis on the Ishigami function

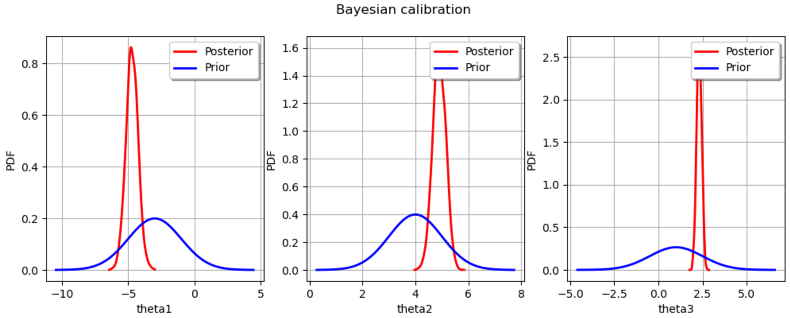
```
estimatorType = ot.HSICVStat()

criticalDomain = ot.Interval(5, float('inf'))
dist2criticalDomain = ot.DistanceToDomainFunction(criticalDomain)
f = ot.SymbolicFunction(["x"], ["exp(-x)"])
filterFunction = ot.ComposedFunction(f, dist2criticalDomain)

TSAEstimator = ot.HSICEstimatorTargetlSensitivity(
    covarianceModelCollection, X, Y, estimatorType,
    filterFunction)

TSAEstimator.run()

graph1 = TSAEstimator.drawHSICIndices()
graph2 = TSAEstimator.drawPValuesAsymptotic()
graph3 = TSAEstimator.drawR2HSICIndices()
graph4 = TSAEstimator.drawPValuesPermutation()
```

Example : Target sensitivity analysis on the Ishigami function



Metropolis-Hastings

- RandomWalkMH now updates with the instrumental in one go
- new Gibbs class that updates components sequentially
- Separate method to define the likelihood
- Handle improper prior via a log-pdf function
- Direct inconditional sampling according to a random variable

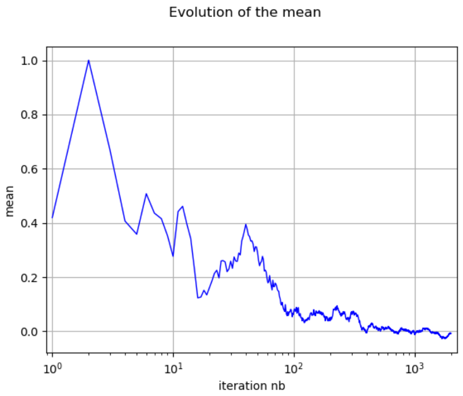


Metropolis-Hastings

```
from openturns import RandomWalkMetropolisHastings as RWMH
rwmh1 = RWMH(prior, initialState, proposal1, [0])
rwmh2 = RWMH(prior, initialState, proposal2, [1, 2])
mh_coll = [rwmh1, rwmh2]
for mh in mh_coll:
    mh.setLikelihood(conditional, y_obs, linkFunction, x_obs)
sampler = ot.Gibbs(mh_coll)
x = sampler.getSample(1000)
```

Iterative statistics

- Compute statistics without the need to store whole samples
- Moments / threshold exceedance / extrema
- Useful in HPC contexts



Iterative statistics

```
iter_mom = ot.IterativeMoments(order, dim)
iter_ext = ot.IterativeExtrema(dim)
iter_te = ot.IterativeThresholdExceedance(dim, 4.0)

for i in range(size):
    x = rv.getRealization()
    iter_mom.increment(x)
    iter_ext.increment(x)
    iter_te.increment(x)

xmin = iter_ext.getMin()
xmax = iter_ext.getMax()
variance = iter_mom.getVariance()
skewness = iter_mom.getSkewness()
pf = iter_te.getThresholdExceedance()
```

NAIS

- Nonparametric Adaptive Importance Sampling (Morio 2015)
- Adaptive method based on the idea of Importance Sampling

$$\widehat{P}^{\mathrm{IS}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{g(\mathbf{x}_i) < T} \frac{h_0(\mathbf{x}_i)}{h(\mathbf{x}_i)}$$

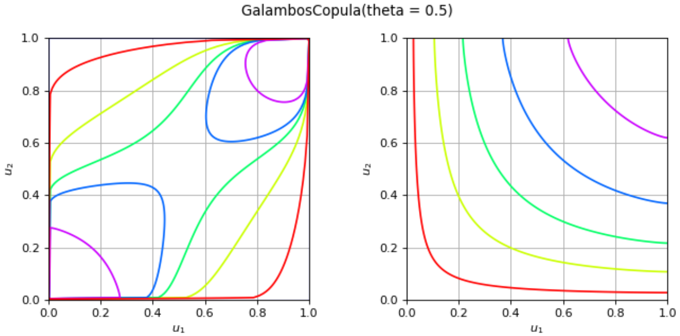
- Updates an importance density from KS (Gauss + Silverman)
- Comparable to subset simulations
- No tuning needed

Galambos copula

- bivariate extreme value copula

$$C(u_1, u_2) = u_1 u_2 \exp\left[(-\log(u_1))^{-\theta} + (-\log(u_2))^{-\theta}\right]^{-1/\theta}$$

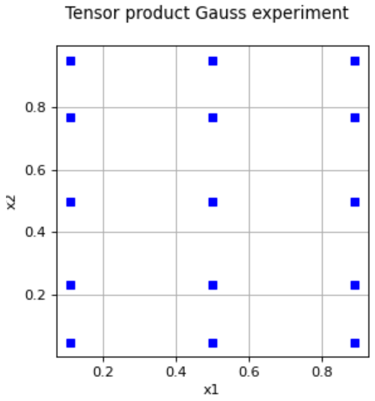
- aims at modelling the dependence of rare events



Tensor product experiment

- Tensorize a set of elementary (d>=1) designs of experiments

$$\int_{\mathcal{X}} g(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} \approx \sum_{i=1}^{s_t} w_i g\left(\boldsymbol{x}_i\right)$$

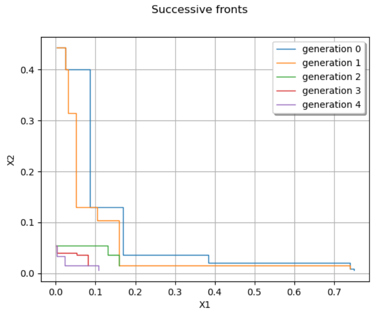


Tensor product experiment

```
exp1 = ot.GaussProductExperiment(ot.Uniform(0.0, 1.0), [3])
exp2 = ot.GaussProductExperiment(ot.Uniform(0.0, 1.0), [5])
exp_tens = ot.TensorProductExperiment([exp1, exp2])
nodes, weights = exp_tens.generateWithWeights()
```

Multi-objective optimization

- massively parallel optimization Pagmo library from ESA
- 18 bio-inspired and evolutionary global algorithms
- 4 multi-objective algorithms
- some support batch evaluation / constraints / MINLP



Multi-objective optimization

```
pop0 = ot.ComposedDistribution([ot.Uniform(0.0, 1.0)] * 2).getSample(100)
algo = ot.Pagmo(zdt1, 'nsga2', pop0)
algo.setGenerationNumber(180)
algo.run()
result = algo.getResult()
pop1 = result.getFinalPoints()
fronts = result.getParetoFrontsIndices()
```

Other improvements

- New maintainer in Debian (P. Gruet, EDF)
- Weekly Python wheels for Windows too
- Continued bugfix / documentation / modules effort

END

Thank you for your attention!
Any questions?