

# OpenTURNS release highlights

Régis Lebrun, Julien Schueller

UserDay #13, 5 June 2020, online event



New features since last year in releases:

- v1.14: 2019-11-13
- v1.15: 2020-05-25

- 1 Probabilistic modelling capabilities
- 2 System events
- 3 HMAT and OpenTURNS: it works (at last!)
- 4 Optimization solvers
- 5 Miscellaneous

# New distributions

- SquaredNormal
- DiscreteCompoundDistribution
- WeibullMax
- Pareto
- MixedHistogramUserDefined



# New estimators

- ParetoFactory
- GeneralizedExtremeValueFactory
- PlackettCopulaFactory
- LeastSquaresDistributionFactory (generic service)



# New copulas

- JoeCopula (EV)
- MarshallOlkinCopula
- PlackettCopula



# System events

- IntersectionEvent:  $E_{sys} = \bigcap_{i=1}^N E_i$

- UnionEvent:  $E_{sys} = \bigcup_{i=1}^N E_i$

- MultiFORM:

$$P(\bigcup_{i=1}^N E_i) = 1 - \Phi_k(\underline{\underline{\beta}}; \underline{\underline{\rho}})$$

- SystemFORM:

$$P(\bigcup_{i=1}^N E_i) = 1 - \Phi_k(\underline{\underline{\beta}}; \underline{\underline{\rho}})$$

$$P(\bigcap_{i=1}^N E_i) = \Phi_k(-\underline{\underline{\beta}}; \underline{\underline{\rho}})$$

$$P(E_{sys}) = P(\bigcup_{i=1}^N E_i) =$$

$$\sum_{i=1}^N P(E_i) - \sum_{i < j} P(E_i \cap E_j) + \dots + (-1)^N P(E_1 \cap E_2 \cap \dots \cap E_N)$$



## ● Modelisation

```
X = ot.RandomVector(ot.Normal(dim))
f1 = ot.SymbolicFunction(['x1', 'x2'], ['(x1+2*x2)^2'])
f2 = ot.SymbolicFunction(['x1', 'x2'], ['(x2+3*x1)^2'])
v1 = ot.CompositeRandomVector(f1, X)
v2 = ot.CompositeRandomVector(f2, X)
e1 = ot.ThresholdEvent(v1, ot.Greater(), 5.0)
e2 = ot.ThresholdEvent(v2, ot.Greater(), 5.0)

# define union/intersection
e3 = ot.IntersectionEvent([e1, e2])
e4 = ot.UnionEvent([e1, e2])
e5 = ot.UnionEvent([e3, e4])

# sample system events
e3.getSample(10)
algo = ot.SubsetSampling(e4)
```



- Algorithms

```
# System-FORM
e5 = ot.IntersectionEvent([e1, e2])
e6 = ot.IntersectionEvent([e3, e4])
e7 = ot.UnionEvent([e5, e6])
solver = ot.AbdoRackwitz()
starting_pt = [0.1] * dim
algo = ot.SystemFORM(e7, event, starting_pt)
algo.run()
result = algo.getResult()
pf = result.getEventProbability()
```

# Discretization of covariance models on large meshes I

There are many contexts in which the discretization of a covariance model over a large mesh is mandatory:

- to sample a Gaussian process on a large mesh
- to build a Kriging meta-model from a large dataset
- to build the Karhunen-Loeve decomposition of a known covariance model on a large mesh
- ...

The resulting matrices are **dense** and **large**, so the first limit is the  $\mathcal{O}(N^2)$  memory need ( $\sim 30000$  nodes for covariance model of dimension 3 with 64GB of memory).

Using HMAT, the memory footprint is reduced to  $\mathcal{O}(N \log N)$ , allowing for much larger meshes (tested on a 900720 nodes mesh with for a covariance model of dimension 3 with 64GB of memory).

# Discretization of covariance models on large meshes II

Then, the factorization times, which is  $\mathcal{O}(N^3)$  with a dense solver (LAPACK) becomes the limiting factor. With HMAT, it drops to  $\mathcal{O}(N \log^2 N)$ .

These were the motivation for the introduction of HMAT within OpenTURNS 1.4, nearly 6 years ago... and it didn't worked as expected!

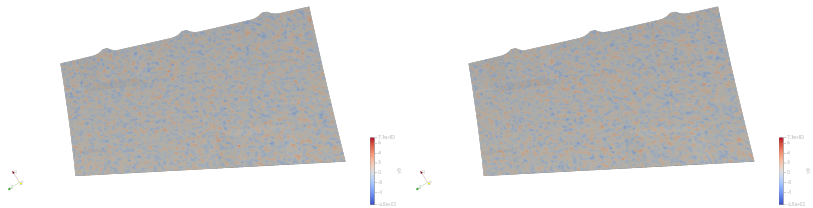
- The regularization procedure developed for LAPACK was not adapted to HMAT, resulting into many (many many) factorizations;
- No compression procedure proposed by the hmat-oss library was adapted to covariance operators.

These points have been solved in two steps, thanks to Romain Poncet's contributions:

- ① OpenTURNS 1.14: the ability to change the regularization factor without recompressing the matrix, and the automatic computation of a reasonable regularization factor (in OpenTURNS 1.14);
- ② OpenTURNS 1.15: The development of the **ACA-random** compression procedure in hmat-oss and its interface in OpenTURNS.

**AND IT WORKS!** Here we compare LAPACK (with OpenBLAS, 4 cores) and hmat-oss (1 core).

# Sampling of a Gaussian process I



**Figure:** Matérn model, 18292 nodes, LAPACK (left, 188 s), HMAT (right, 90 s)

# Sampling of a Gaussian process II



Figure: SquaredExponential model, 900720 nodes, HMAT (610 s)

# Sampling of a Gaussian process III



Figure: Factorization time (left) and sampling time (right), lower is better

The Karhunen-Loeve decomposition of a covariance model on a mesh results in the eigen decomposition of a dense matrix  $M = CG$  where  $G$  is a sparse matrix, the Gram matrix of the  $P_1$  basis over the mesh, and  $C$  a dense, positive symmetric definite matrix, the discretization of the covariance model on the mesh.

- A naive approach is to compute  $M$  and to decompose it, which was the only option up to OpenTURNS 1.14.
- Using an iterative algorithm for the eigen decomposition, the product  $z = Mx$  for a given vector  $v$  can be done using two products  $y = Gx$  and  $z = Cy$ , both allowing for optimizations.



# Karhunen-Loeve decomposition II

We rely on the **Spectra** library (<https://github.com/yixuan/Spectra>) for the iterative algorithm, we use a compressed sparse column compression scheme for  $G$  and an hmat compression for  $C$  (no regularization/factorization step here).

**Bonus:** using **Spectra**, one can select the maximum number of modes to be computed. Here we choose to compute 20 modes.



**Figure:** SquaredExponential model, 2268 nodes, mode 11, LAPACK (left, 15.3 s), HMAT (right, 0.1 s)

# Karhunen-Loeve decomposition III



Figure: SquaredExponential model, 224515 nodes, HMAT (30.1 s)

# Karhunen-Loeve decomposition IV



**Figure:** Decomposition time using LAPACK, Spectra/LAPACK and Spectra/HMAT

# Optimization solvers

- Dlib (continuous, general-purpose: CG, BFGS, LBFGS, Newton, GLOBAL, LSQ, TrustRegion)
- Ipopt (continuous, large dimension)
- Bonmin (mixed, large dimension: NLP BB, Outer-approx decomp, Quesada and Grossmann's BC, Hybrid outer-approx BC, Iterated feasibility pump)



# Performance improvements

- TBB/OpenBLAS parallel regions conflict: huge penalty
- TBB-enabled Windows binaries
- KernelMixture.computePDF x10 speed
- Speed-up Tensor metamodel evaluation: x2
- ODE solvers speedup
- Memory leaks in Python/SWIG layer

- Sample improvements

```
x = ot.Sample([[42.0]*3]*8)

# by description marginal accessors
x.getMarginal(['v1', 'v2'])

# get points #2, #5, #6
x[[2, 5, 6]]

# __contains__ operator
[42]*3 in x
=> True
```

# Other improvements

- Documentation: added examples
- Bugfixes (and reports), lots of them!
- Gitter chat
- New version of otagrum module

Thank you for your attention!  
Any questions?

