

# Low rank tensor approximation in OpenTURNS

**J. Schueller**

Phimeca

CHORUS workshop, 2017-10-19



# Plan

Non-polynomial basis

Canonical tensor

# Fourier series

## Fourier series

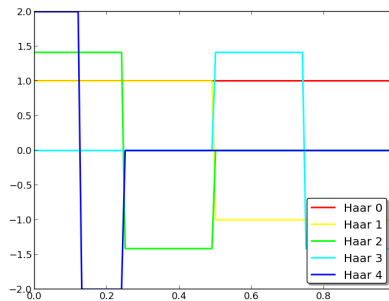
$$\begin{aligned}\psi_0(x) &= 1 \\ \psi_{2k+1}(x) &= \sqrt{2} \sin(kx) \\ \psi_{2k+2}(x) &= \sqrt{2} \cos(kx)\end{aligned}$$



# Haar wavelets

## Haar wavelets

$$\begin{aligned}\psi_0(x) &= \mathbb{1}_{[0,1]}(x) \\ \psi_n(x) &= \frac{1}{2^{j/2}} \left[ \mathbb{1}_{[\frac{k}{2^j}, \frac{k+1/2}{2^j}]}(x) - \mathbb{1}_{[\frac{k+1/2}{2^j}, \frac{k+1}{2^j}]}(x) \right]\end{aligned}$$



# Usage

## In Python...

```
# as a regular function
family = ot.FourierSeriesFactory()
family = ot.HaarWaveletFactory()

for i in range(5):
    f = family.build(i)
    d = f.draw(xmin, xmax, 100)
```

# Functional chaos tensorization

## Tensorization

Functional chaos decomposition

$$Y \equiv h(\underline{X}) = \sum_{j=0}^{\infty} a_j \psi_j(\underline{X})$$

on univariate orthogonal function families

$$\phi_1^{(j)}, \dots, \phi_M^{(j)} \quad \forall j \in [1, d]$$

upon tensorized basis

$$\psi_{\underline{\alpha}}(\underline{x}) \equiv \phi_{\alpha_1}^{(1)}(x_1) \times \dots \times \phi_{\alpha_d}^{(d)}(x_d)$$

# Functional chaos tensorization

## Tensorization

upon tensorized basis

$$\psi_{\underline{\alpha}}(\underline{x}) \equiv \phi_{\alpha_1}^{(1)}(x_1) \times \cdots \times \phi_{\alpha_d}^{(d)}(x_d)$$

multi-indices notation

$$\alpha \equiv \{\alpha_1, \dots, \alpha_d\}$$

curse of dimensionality

$$P = C_M^{d+M}$$



# Usage

## In Python...

```
# polynomial basis
ef = ot.LinearEnumerateFunction(dim)
factC = [LegendreFactory()] * dim
prod = ot.OrthogonalProductPolynomialFactory(factC, ef)

# non-polynomial basis
factC = [ot.FourierSeriesFactory()] * dim
prod = ot.OrthogonalProductFunctionFactory(factC)

algo = ot.FunctionalChaosAlgorithm(...)
```



# Rank one tensor

## Rank one tensor

$$f(x_1, \dots, x_d) = \prod_{i=1}^d v_i(x_i)$$

with

$$v_i = \sum_{j=1}^{n_i} \alpha_j^{(i)} \phi_j(x_i)$$

expanding to

$$\begin{aligned} f(x_1, \dots, x_d) &= (\alpha_1^{(1)} \phi_1(x_1) + \dots + \alpha_{n_1}^{(1)} \phi_{n_1}(x_1)) \\ &\quad \times \dots \\ &\quad \times (\alpha_1^{(d)} \phi_1(x_d) + \dots + \alpha_{n_d}^{(d)} \phi_{n_d}(x_d)) \end{aligned}$$

## Canonical tensor format

Available representation

$$f(x_1, \dots, x_d) = \sum_{k=1}^r \prod_{i=1}^d v_i^{(k)}(x_i)$$

with

$$v_i = \sum_{j=1}^{n_i^{(k)}} \alpha_j^{(i,k)} \phi_j(x_i)$$

linear number of terms wrt dimension

$$P = r \sum_{i=1}^d n_i$$

# Alternating Least Squares

## Alternating Least Squares algorithm

Allows to learn a rank-one tensor.

---

### Algorithm 1 ALS

---

- 1: Initialize  $v_i(x_i) = 1$
  - 2: **while**  $v$  does not converge **do**
  - 3:   **for**  $i = 1$  to  $d$  **do**
  - 4:      $[\Psi^i(x)]_j = \prod_{u=1 \neq i}^d v_u(x_u) \phi_j^i(x_i)$
  - 5:     Solve  $\operatorname{argmin}_{\beta_i} \|y - \Psi^i(x)^t \beta_i\|_2^2$
  - 6:   **end for**
  - 7: **end while**
-

# Greedy rank-one approximation

## Alternating Least Squares algorithm

Allows to learn a rank- $r$  tensor.

---

### Algorithm 2 Greedy rank-one

---

- 1: Rank-1 approximation  $\prod_{i=1}^d v_i^{(1)}(x_i)$
  - 2: **for**  $r = 2$  to  $r_{max}$  **do**
  - 3:   Rank-1 approximation  $\prod_{i=1}^d v_i^{(r)}(x_i)$
  - 4:    $y^m = y - \sum_{k=1}^r \alpha_k \prod_{i=1}^d v_i^{(k)}(x_i)$
  - 5:   Update  $\alpha$  to minimize error (least-squares)
  - 6: **end for**
-

# Usage

## In Python...

```
import openturns as ot

def model(X):
    ...
    return Y

r = 4
dim = 5
nk = [10] * dim
factC = [ot.FourierSeriesFactory()] * dim
prod = ot.OrthogonalProductFunctionFactory(factC)
X = dist.getSample(size)
Y = model(X)
algo = ot.TensorApproximationAlgorithm(X, Y, dist, factC, nk, r)
```

# Usage

## In Python...

```
algo = ot.TensorApproximationAlgorithm(X, Y, dist, factC, nk, r)
algo.rvn()
```

# Conclusion

## Conclusions

1. Greedy rank-1
2. Regularized greedy rank-one

## Perspectives

- ▶ Rank-M