Low-rank in Uncertainty Management

Efficient linear algebra and an overview of the $\mathcal{H} -$ *matrix* framework

Kieran Delamotte

IMACS

May, 2021

First encounter with low rank

Key ideas for fast computation

Uncertainty quantification (UQ) typically requires a high number of simulations using basic linear algebra operations. Then two things come in handy :

- fast linear algebra decomposition (QR, LL^T, ...)
- efficient data strucutures

If many simulations are 'similar' there is most likely a low-rank property somewhere to be exploited. Low-rank property in linear algebra leads to fast computations AND efficient structures !
We shall present two efficient methods :

- random linear algebra for medium-sized problems (on a simple example !)
- the $\mathcal{H}$-matrix framework (much more detailed)

Model problem

Suppose we are interested in eigenvalues (and/or eigenvectors) of a covariance matrix of the form $C_X = X^T X$ where X is of size $m \times n$. Several methods :

- eigenvalue decomposition of C_X : $\mathcal{O}(n^3)$ operations and conditionning of normal equations is bad !
- SVD decomposition of X : (better, still expensive) $\mathcal{O}(mn^2)$ operations
- what else ?

Randomness and linear algebra

Basic idea : using random test matrix Ω and the fact that any orthogonal basis of $\mathrm{ran}(Y)$ with $Y = A\Omega$ is a good approximate basis for $\mathrm{ran}(A)$. Y has a less columns than A so QR is cheaper !

Algorithm 1 A simple random QR decomposition

Require: A matrix $\mathbf{A}$ of size $m \times n$, an oversampling parameter l.
Ensure: An orthogonal basis $\mathbf{Q_Y}$ of $\mathrm{ran}(\mathbf{A})$.

1: Draw a random matrix $\mathbf{W}$ of size $n \times l$.
2: Form product $\mathbf{Y}$: $\mathbf{Y} = \mathbf{AW}$.
3: Form the QR decomposition of the matrix $\mathbf{Y}$: $\mathbf{Y} = \mathbf{Q_Y R}$.
4: **return**

A random SVD

Algorithm 2 A simple random SVD decomposition

Require: A matrix $\mathbf{A}$ of size $m \times n$, an orthogonal matrix $\mathbf{Q}$ tq $\mathbf{A} \simeq \mathbf{QR}$.

Ensure: An approximate *SVD* decomposition of $\mathbf{A}$, $\mathbf{A} \simeq \mathbf{U\Sigma V}^T$.

Construct the projection matrix $\mathbf{B} = \mathbf{Q}^T\mathbf{A}$.

2: Form the SVD of $\mathbf{B}$: $\mathbf{B} = \tilde{\mathbf{U}}\mathbf{\Sigma V}^T$.

Build the matrix $\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$.

4: **return**

A numerical example

Say $X \in \mathbb{R}^{m\times n}$ is a discrete approximate of a random process $X(t,\omega)$ over $[-1,1]$ with exponential covariance function $C(s,t) = e^{-|t-s|}$. We are still interested in eigenvalues of C_X. In that case, we can prove that $\lambda_n/\lambda_1 = \mathcal{O}(n^{-2})$.
In the following example we choose $m = 6400$ and $n = 1000$.

Eigenvalues estimates

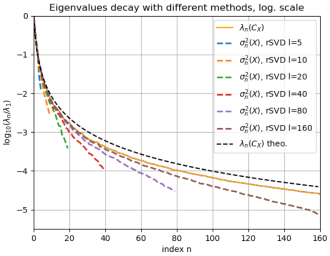


FIGURE – Eigenvalues decay.

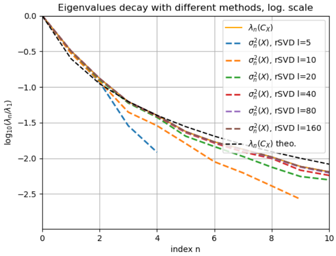


FIGURE – Eigenvalues decay - the first 10.

Case summary

test matrix # cols.	elapsed time (s)	$\sum \lambda_n$	rel. error
5	0.021	4950345.8	0.253
10	0.027	5894250.4	0.111
20	0.038	6309314.4	0.048
40	0.060	6478969.3	0.023
80	0.068	6556588.8	0.011
160	0.131	6598333.6	0.005

TABLE – Example summary for random SVD

- For reference : $\mathrm{trace}(C_X) = 6632269.0$
- elapsed time for full SVD : 0.663s
- elapsed time for *numpy.linalg.eigs* : 107.307.

Partial conclusion

- many articles in the literature,
- random SVD is implemented in OPENTURNS,
- generally behave better than classical methods,
- what if the problem is larger ?

First encounter with low rank

Large and dense systems

$\mathcal{H}-matrices$

Applications

Conclusions & perspectives

Introduction

Several applications in statistics lead to large and dense matrices :

- Karhunen-Loève decomposition ;
- Kriging ;
- applications using large covariance matrices (ex : random Gaussian sampling).

Covariance matrices

Let $Z : \mathbb{R}^d \mapsto \mathbb{R}$ be some random process, stationnary of order 2. We have N observation points $\left\{x_i \in \mathbb{R}^d / i = 1, \dots, N\right\}$ with values $Z(x_i)$. Let assume that the covariance of these points is known and given by a matrix $K \in \mathbb{R}^{N \times N}$ with

$$K_{ij} = \mathrm{Cov}(Z(x_i), Z(x_j))$$

Covariance matrices

covariance kernel

- usually K is not known exactly :
  - modelled as a convolution matrix of a kernel $k : \mathbb{R}^d \mapsto \mathbb{R}_+$:

$$K_{ij} := k(x_i, x_j)$$

  - what is k ?
    - Exponential :

$$k(x,y) = e^{-|x-y|/\lambda}$$

    - Gaussian :

$$k(x,y) = e^{-|x-y|^2/(2\lambda^2)}$$

    - Quadratic :

$$k(x,y) = \left(1 + \frac{|x-y|}{2\lambda}\right)^{-2}$$

    - and many more !
- N can be large. For instance, every node in a FEM discretization ;

Covariance matrices

linear algebra

- K is ill-conditionned, many RHS : direct solver ;
- K is SPD : Cholesky.

Complexity (LAPACK)

- Cholesky factorization (DPOTRF) : $1/3N^3 + 1/2N^2 + 1/6N$
- Solving (DPOTRS) : $N_{\mathrm{RHS}} \times 2N^2$
- Storage : $8N^2/2$

Need of a fast direct solver : $\mathcal{H}$-matrix framework.

Toy model : exponential kernel in 1D

Let X_N be a uniform discretization of $[0, 1]$ with the discretization step $h = \frac{1}{N-1}$:

$$X_N = \{0 = x_1, \ldots, x_N = 1\}$$

The correlation length λ is set to $\lambda := 5h$ and the exponential kernel in 1D reads as

$$K_\lambda(x_i, x_j) = e^{-|x_i - x_j|/\lambda}$$

Toy model : exponential kernel in 1D

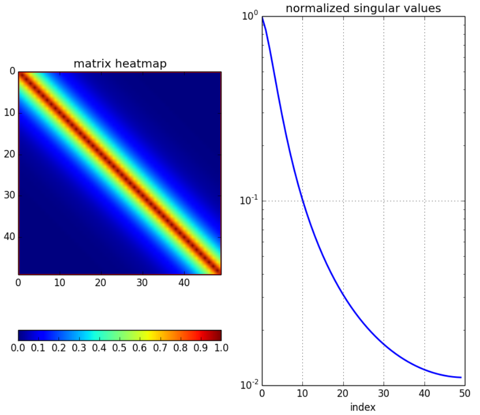


FIGURE – the covariance matrix $K_\lambda([0,1],[0,1])$

Toy model : exponential kernel in 1D

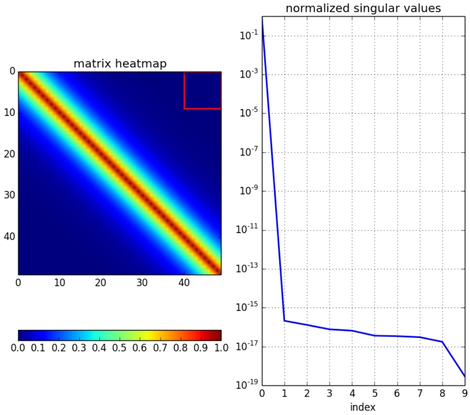


FIGURE – small extra-diagonal block : $K_\lambda([0, 0.2], [0.8, 1])$

Toy model : exponential kernel in 1D

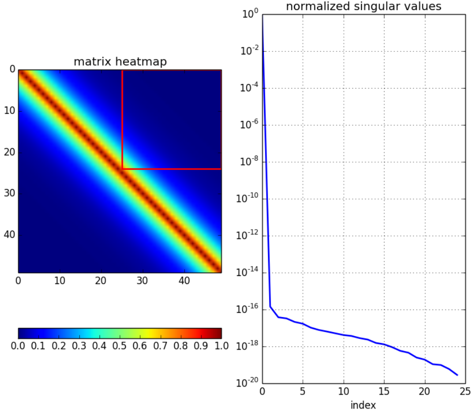


FIGURE – large extra-diagonal block : $K_\lambda([0, 0.5], [0.5, 1])$

Toy model : exponential kernel in 1D

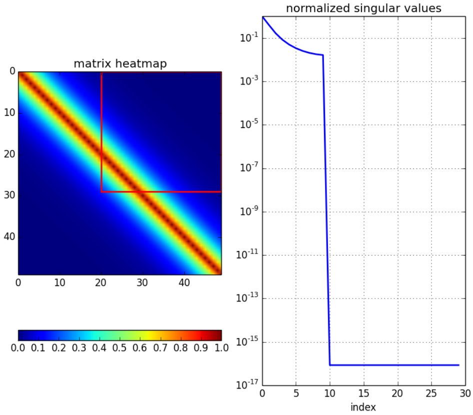


FIGURE – taking a part of the diagonal : $K_\lambda([0, 0.6], [0.4, 1])$

Toy model : exponential kernel in 1D

Whenever x_i and x_j are in disjoint sets the kernel reads as a separated one. For instance ; if $x_i > x_j$ then

$$K_\lambda(x_i, x_j) = e^{(x_j - x_i)/\lambda} = e^{x_j/\sqrt{\lambda}} e^{-x_i/\sqrt{\lambda}},$$

which is of rank 1.

First encounter with low rank

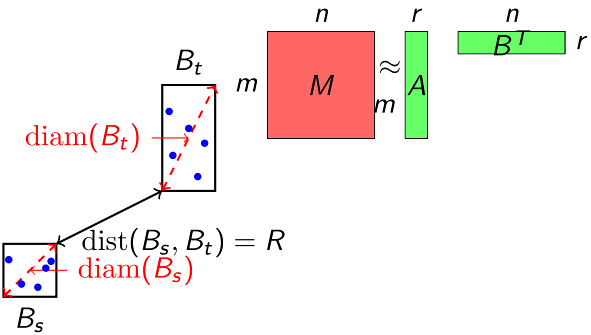
Large and dense systems

$\mathcal{H}-$ *matrices*

Applications

Conclusions & perspectives

Admissibility gives low-rank



Usual condition :

$$\min(\mathrm{diam}(B_t), \mathrm{diam}(B_s)) \leqslant \eta\, \mathrm{dist}(B_t, B_s) \qquad \text{(admissibility condition)}$$

The separation condition $R = 0$ gives the **HODLR**(**H**ierarchically **O**ff-**D**iagonal **L**ow-**R**ank) structure described by the 1D toy model.

Low-rank approximation : compression techniques

- SVD : $M \approx U \Sigma V^H$
  - Rank and precision controlled ;
  - Costly $\mathcal{O}(4m^2n + 8mn^2 + 9n^3)$ (hyp : $m > n$)
- Existence of cross approximations : row/col. extraction (Goreinov & Tyrtyshnikov '97) ;
- Gaussian/LU rank-revealing scheme known as **Full Cross Approximations** :

1: **while** $\|M\| \geq \varepsilon \|M_0\|$: **do**
2: $\quad \mathrm{rank}(M) \leftarrow \mathrm{rank}(M) + 1$
3: $\quad$ Find the coefficient $M_{i^\star j^\star}$ so that $M_{i^\star j^\star} = \max_{i,j} |M_{ij}|$, $\alpha = M_{i^\star j^\star}$
4: $\quad M \leftarrow M - \frac{1}{\alpha} M(:, j^\star) M(i^\star, :)$
5: **end while**

Variants

The fast determination of the pivot is the main idea of all fast algorithms. Key points to speed up the full cross approximation :

- **Partially pivoted Cross Approximation**.
  - We seek the largest pivot over a column and/or a row in $\mathcal{O}(m)$ instead of $\mathcal{O}(m^2)$ operations.
  - Only the modified coefficients of the remainder are computed at each step.
- **Adaptive CA algorithm** : a fast (linear) estimation of the remainder.
- **ACA+** and other variants use other heuristics.
- Trade-off between robustness (SVD) and efficiency (ACA/ACA+) : computations from $\mathcal{O}(m^2n + mn^2)$(SVD) to $\mathcal{O}(mnr)$(fullCA) to $\mathcal{O}((m+n)r^2)$(ACA).

Space partitioning : clustering

- Use of **bounding boxes** : easier to handle than point clouds ;
- Recursive splitting strategy (**Divide and Conquer strategy**) thanks to **nested bisection** :
  - geometric : the box is split in two halves along the largest axis ;
  - median : each half contains roughly the same number of unknowns ;
  - others (PCA,...).
- Split the boxes until each box contains a fixed small number of unknowns ;
- Result : **binary tree**.

Space partitioning : clustering

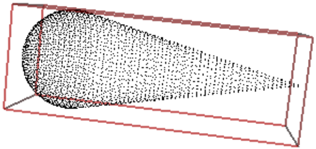


FIGURE – Illustration of the geometric clustering on a cone-sphere.

Space partitioning : clustering

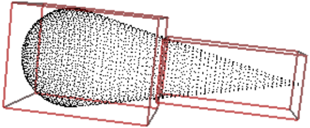


FIGURE – Illustration of the geometric clustering on a cone-sphere.

Space partitioning : clustering

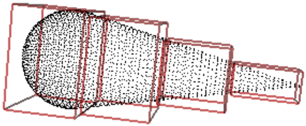


FIGURE – Illustration of the geometric clustering on a cone-sphere.

Space partitioning : clustering

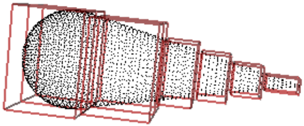
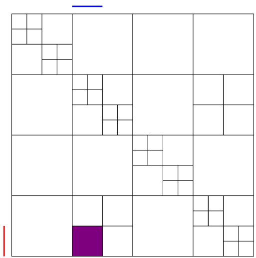


FIGURE – Illustration of the geometric clustering on a cone-sphere.

Blockclustering : Clustering & Admissibility

It is a quad-tree whose nodes are matrix blocks and the leaves are admissible (or small) blocks ; a block is split in a 2×2 block structure.



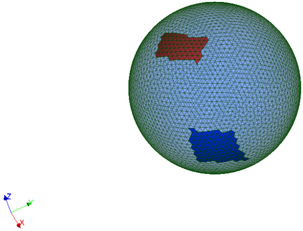


TABLE – Blockclustering and the geometry

The $\mathcal{H}$-matrix structure

A $\mathcal{H}$-matrix is a **quadtree** (with Binary Space Partitioning) :

- Internal nodes : subdivided $\mathcal{H}$-matrix ;
- Leaves :
  - admissible block : large & low-rank ;
  - inadmissible block : dense & full rank, but small.

Remarks

- Only the leaves carry data ;
- Big admissible blocks ($10^4 \times 10^4$ and more)
- Small (and few) inadmissible blocks (100×100).

Each admissible block is compressed with a fast method thus determining a numerical rank with a prescribed relative error ε.

Operations : three kinds

- $\mathcal{H}$ -**BLAS1&2 :** Assembly, AXPY, GEMV
  Simple. Operating only on leaves.
- $\mathcal{H}$ -**BLAS3 :** GEMM, TRSV
  More involved. Operations at many different levels of the same sub-tree.
- $\mathcal{H}$ -**LAPACK :** Inverse, LU, LL^T.
  Uses BLAS2 and BLAS3 operations, harder to implement in parallel.

Base operations

All operations use BLAS/LAPACK. Typical subroutines are : SVD, QR, LU, TRSV, GEMM and GEMV.

Computation details

Typical issues

The $\mathcal{H}$-matrix algorithms are not nice for the hardware :

- Very small operations ;
- Oddly-shaped matrices : "Tall & skinny" ;
- High memory band.

Observations

- Most (70-80%) of the time spent in :
  - QR decompositions of T&S matrices ;
  - SVD decompositions of small matrices.
- BLAS implementions cannot reach the peak performance ;
- Very high memory bandwidth.

Complexity estimates

Useful pointers

- Most complexity estimates assume a fixed upper bound k for the low-rank matrices involved ;
- The structure of the matrix (as represented by a tree) is important as well : a large depth with small blocks is typically a bad omen.

Common operations

For a matrix size of $N \times N$ with the previous assumptions :

- assembly : time and storage is in $\mathcal{O}(kN \log N)$;
- addition : $\mathcal{O}(k^2 N \log N)$ operations ;
- multiplication and Cholesky factorisation : $\mathcal{O}(k^3 N \log^3 N)$ operations ;
- in practice a $\mathcal{O}(N \log^2 N)$ complexity is observed.

Recall the 1D toy model ?

- exponential kernel : $K(s,t) = e^{-|s-t|/\lambda}$
- prescribed relative error $\varepsilon = 10^{-4}$;
- **HODLR** admissibility ;
- 1D exponential kernel : provably rank-one when **HODLR**.
- 'exact model' is the second Hermite function $\psi_2(x) = (2x^2 - 1)e^{-\frac{1}{2}x}$;
- input data as a gaussian distribution ;
- here : one iteration of optimization loop, correlation length $\lambda = 0.01$.

Recall the 1D toy model ?

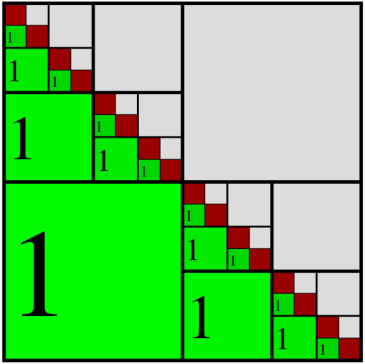


FIGURE – Lower part of the covariance matrix : rank map.

- matrix size 1000×1000 ;
- compression ratio : $\approx 12\%$ (small case !)

Recall the 1D toy model ?

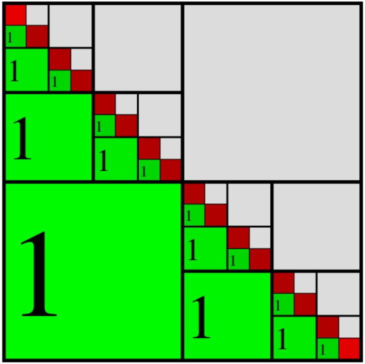


FIGURE – Cholesky factor : rank map.

- matrix size 1000×1000 ;
- compression ratio : $\approx 12\%$ (small case !)

Recall the 1D toy model ?

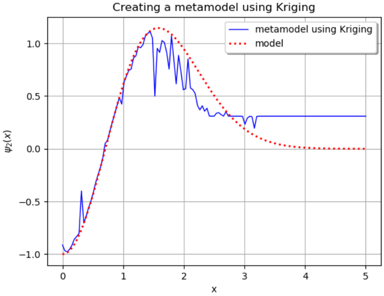
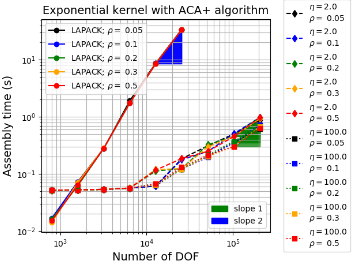


FIGURE – Surface response using kriging

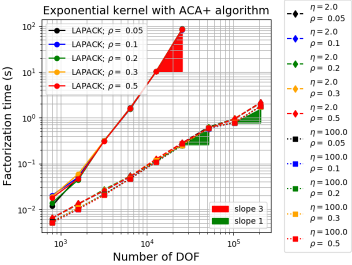
- compression ratio : $\approx 12\%$ (small case !) ;
- same response with LAPACK and HMAT solver ;
- maximum absolute error between two approximates : 1.55×10^{-15}.

Performances - computational time

the exponential kernel in 1D



(a) Matrix assembly

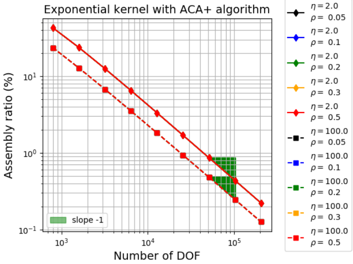


(b) Matrix factorization

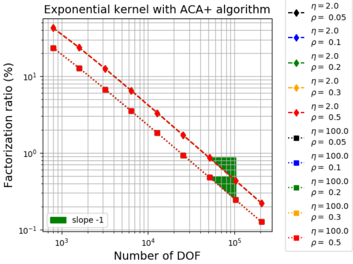
FIGURE – Computational time vs matrix size

Performances - memory

the exponential kernel in 1D



(a) Matrix assembly



(b) Matrix factorization

FIGURE – Memory vs matrix size

3D exemple : a cantilever beam

- The vertical deflection y of a cantilever beam's free end of fixed length L reads as

$$y = \frac{FL^3}{3EI},$$

  where :
  - E is the Young modulus ;
  - F is the load ;
  - I is the moment of inertia.
- *Input* variables $x = (F, E, I)$ are assumed random ;
- Variable of interest (*output*) is the deflection y estimated thanks to the model $\mathcal{M}$

$$\mathcal{M} : x \mapsto y$$

- Building a metamodel $\tilde{\mathcal{M}}$ through Kriging and optimization loop for parameters : one $\mathcal{H}$-matrix at each iteration to treat the covariance matrix associated with a specified covariance model.

The chosen covariance model $K_\lambda(s,t)$ is the following tensor product :

$$e^{-(s_1-t_1)/\lambda_1} e^{-(s_2-t_2)/\lambda_2} e^{-(s_3-t_3)/\lambda_3}.$$

- Let $\lambda_1 = 3.96528$, $\lambda_2 = 5.8237$ and $\lambda_3 = 9.0679$ be the starting coefficients for the optimization.
- Degrees of freedom to be clustered within the $\mathcal{H}$-matrix framework :

```
"Young modulus";"Load";"Inertia"
3.6258375026+07;4.797336026+04;3.5171332517+02
3.1382130227+07;3.350317520+04;3.5324327901+02
... ... ... ... ... ... ... ... ... ... ... ..
```

Results

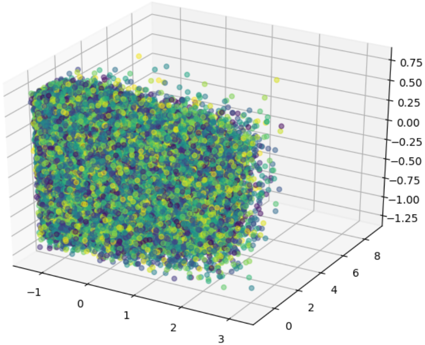


FIGURE – Input data : 5×10^4 entries.

- covariance matrix of size $5.10^4 \times 5.10^4$, symmetric and double precision : full size in memory is 10Go.

Results

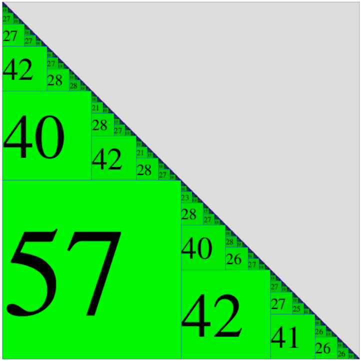


FIGURE – Lower part of the covariance matrix : rank map.

- memory : 167Mo (compression ratio : 1.67%) ;
- assembly time : 11.83s

Results

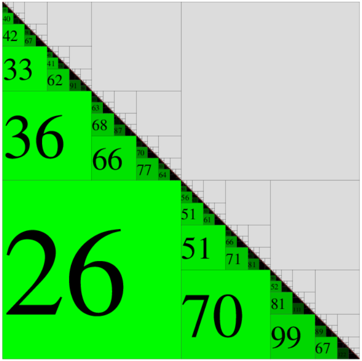


FIGURE – Cholesky factor : rank map.

- memory : 263Mo (compression ratio : 2.63%) ;
- Cholesky time : 17.84s

Gaussian process

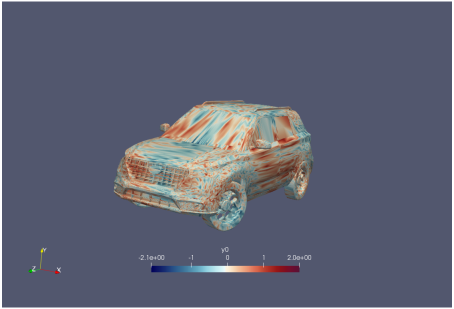


FIGURE – A Gaussian process on a Hyundai FEM mesh

- 144966 vertices ;
- squared exponential covariance kernel.
- Build, Cholesky factorization and trajectory generation in 118s.

Summing up the $\mathcal{H}$-matrices method

- Three key components for assembling a $\mathcal{H}$-matrix :
  - The clustering of degrees of freedom (e.g. geometric) ;
  - An admissibility condition = which block to compress ;
  - A fast on-the-fly algorithm to assembly low-rank admissible blocks.
- An algebra on $\mathcal{H}$-matrices :
  - Multiplications and additions of $\mathcal{H}$-matrices ;
  - Fast factorization of an $\mathcal{H}$-matrix with the same structure :
    - Fast direct solver ;
    - Good preconditioner for iterative solver.

Conclusion

- The $\mathcal{H}$-matrix framework is an enabler for many statistics problems ;
- Sequential solver is freely available through OpenTurns ;
- Efficient parallel solver through licensing (contact Airbus & Imacs).