Olivier Truffinet

Supervisors: Karim Ammar, Bertrand Bouriquet

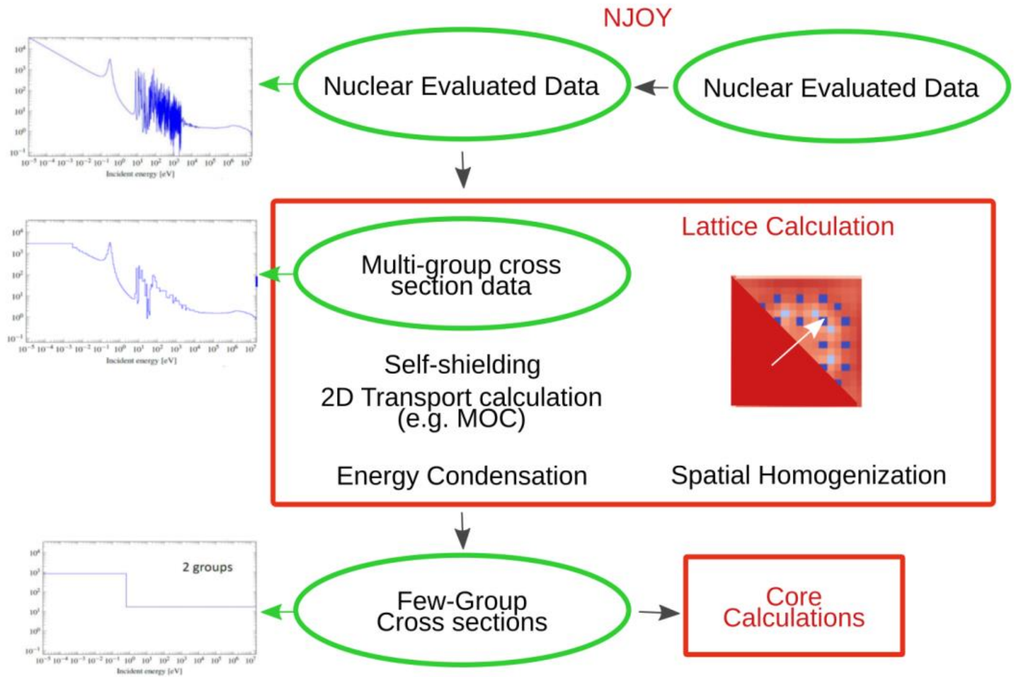
DE LA RECHERCHE À L'INDUSTRIE

Extrapolation and compression of homogenized cross-sections by the EIM method

Sommaire

Introduction

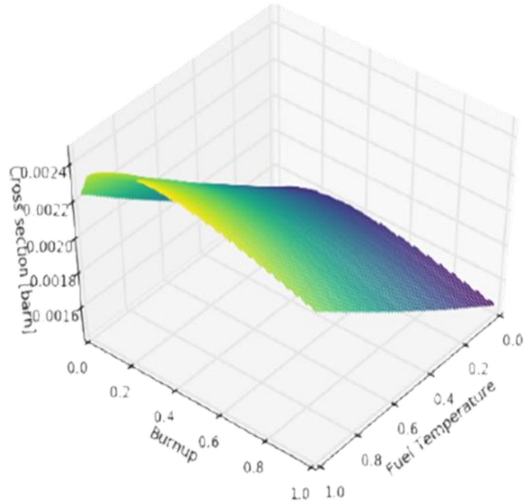
Deterministic codes for neutronics calculation

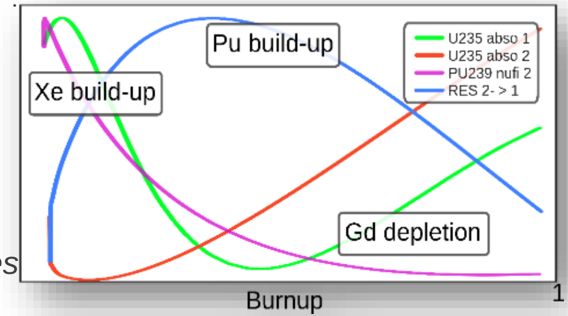


Source: E.Szames, *Few group cross section modeling by machine learning for nuclear reactor*

Homogenized cross-sections

- **Problem** : creating and storing a large number (up to millions) of <u>(strongly) correlated</u> multivariate surrogate models, one for each cross-section.
- Double constraint : very high precision ($\sim 10^{-3}$ relative error), low memory footprint
- Characteristics of the data:
  - Very little noise
  - Smooth, often monotonic
  - Low polynomial order with respect to most variables, one variable more erratic (burnup)

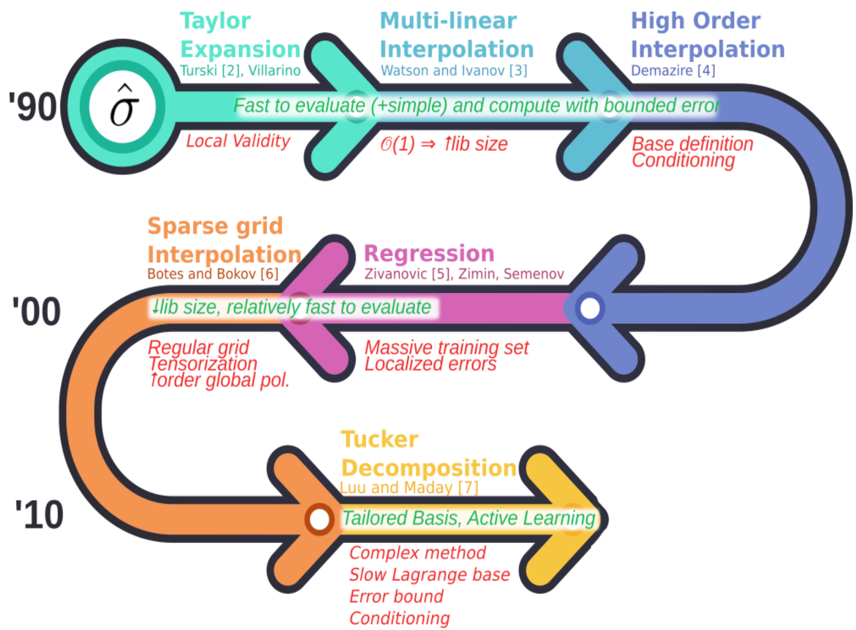




Source: E.Szames

State of the art

- Most common method: multilinear interpolation on tabulated values
- Strengths: simplicity, computation time, error control
- Problems: accuracy, number of coefficients, curse of dimensionality
- Consequence: the size of cross-sections libraries of is exploding (several hundreds of Gb)!



Source: E.Szames, *Few group cross section modeling by machine learning for nuclear reactor*

A small side-step

- **…** I'm not going to adress this full problem today
- Instead, I'm trying to reduce the number of required tabulated values (a.k.a simulation calls) by *extrapolating* some of them

(here, extrapolating means transfering information from some cross-sections to others, though they're only statistically correlated)

- Other objective : compressing the tabulated data
- Topics at play : **greedy approximation, decomposition on snapshots, optimal sampling**

Empirical Interpolation Method (EIM)

Empirical interpolation method (EIM)

- Greedy algorithm originally designed for approximation of nonlinear parametric functions in reduced basis methods for PDEs

- Base-coefficients decomposition : estimator $\hat{f}(\vec{x},\mu) = \sum_{i=1}^{r} c(\mu) B(\vec{x}) \approx f(\vec{x},\mu)$

Interpolation coefficients

→ Matrix form : $\boldsymbol{F} \approx \boldsymbol{C} \cdot \boldsymbol{B}$ (standard matrix decomposition…)

- «Interpolation» means «computing coefficients $c(\mu)$ from r samples ».
  !! It's an entirely discrete process !!

- It is done by inverting the interpolation system : $\hat{f}\left(\vec{x}_{\rho_i},\mu\right) = f\left(\vec{x}_{\rho_i},\mu\right) \forall\, i \in [\![1;r]\!]$, where the ρ_i are the interpolation points

→ Matrix form : $\boldsymbol{C} = \boldsymbol{F}\boldsymbol{P}_{\rho}(\boldsymbol{B}\boldsymbol{P}_{\rho})^{-1}$, with $\boldsymbol{P}_{\rho}$ projection matrix on the interpolation points

Application to matrix compression

- We replace "values of a parametric function" by "matrix with strongly correlated lines“ (i.e μ can be a discrete label). We're still looking for a decomposition $\boldsymbol{F} \approx \boldsymbol{C} \cdot \boldsymbol{B}$

- Interest for matrix compression: the factorized form contains only $nr + rp = r(n+p)$ floating numbers (avec r$\ll n, p$) instead of np for the original matrix

- Compression rate : $\mathcal{R} = \frac{np}{r(n+p)} \in \left[\frac{\min(n,p)}{2r}, \frac{\min(n,p)}{r}\right]$

→Linear structure ⇒ very fast decompression, commutative with some operations (linear interpolation, linear combinations, slicing…)

Application to extrapolation

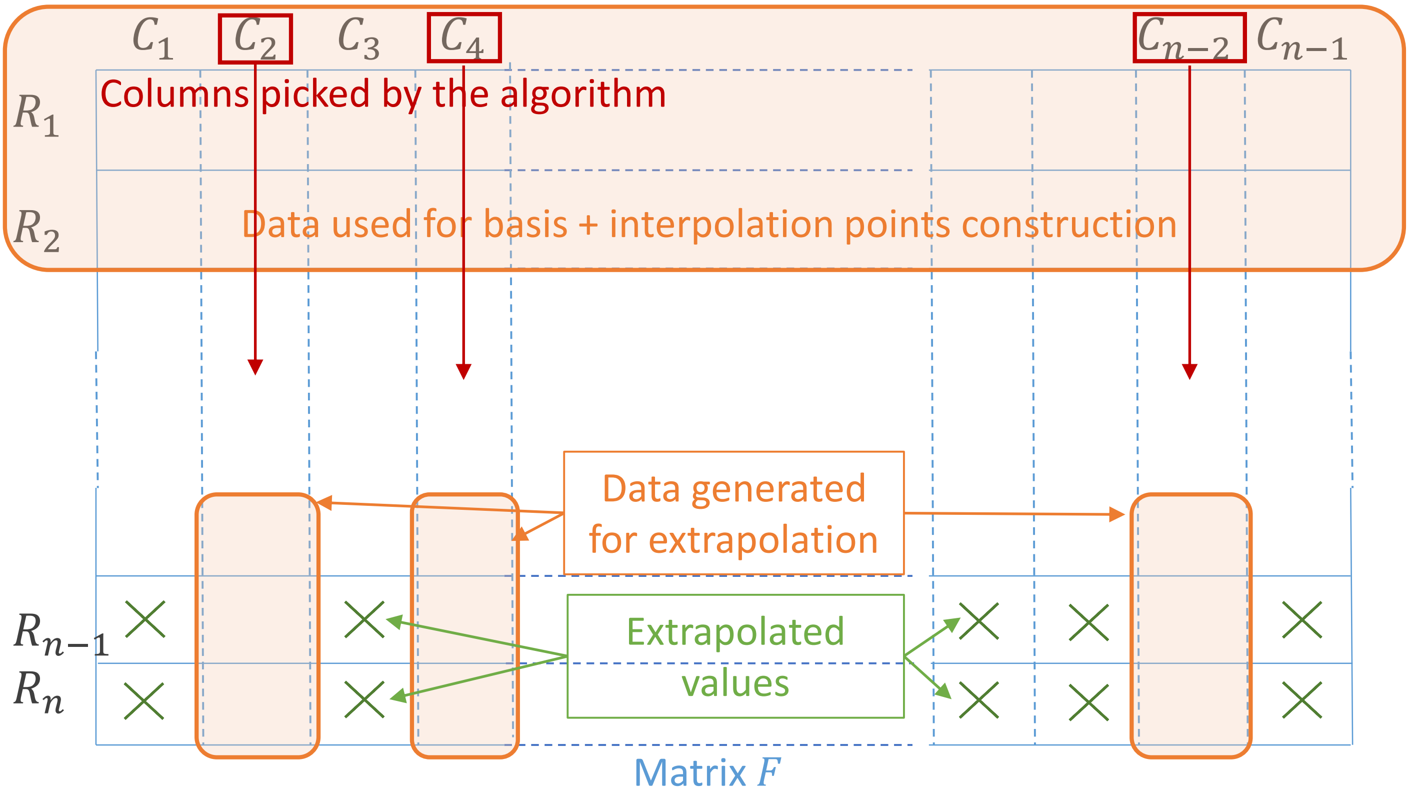
- Matrix rows are outputs of a physics code ; columns correspond to (flattened) multi-parametrization. We assume that the matrix can be generated column after column (*active learning*)

- Let $\boldsymbol{f} \epsilon \mathbb{R}^p$ be a matrix row, $\tilde{\boldsymbol{f}} \epsilon \mathbb{R}^r$ its values at the points $\rho_1, \dots, \rho_r$. We can interpolate the compressed coefficients of $\boldsymbol{f}$ by $\mathbf{c} = \tilde{\boldsymbol{f}}(\boldsymbol{B}\boldsymbol{P}_\rho)^{-1}$, then reconstruct the full vector $\boldsymbol{f}$: $\boldsymbol{f} \approx \mathbf{c} \cdot \boldsymbol{B}$

- In other words: **if we already have a base and interpolation points, we can generate the code outputs at the points ρ_i only, and interpolate all the remaining values**

/!\ This “interpolation” is discrete, only aimed to spare some computation : /!\
we cannot say anything outside of the predifined support points !

Application to extrapolation



Basis construction

- Greedy algorithm based on the infinite norm
- At each step, we add the line of data least well reproduced by the current model, and the point responsible for the largest error

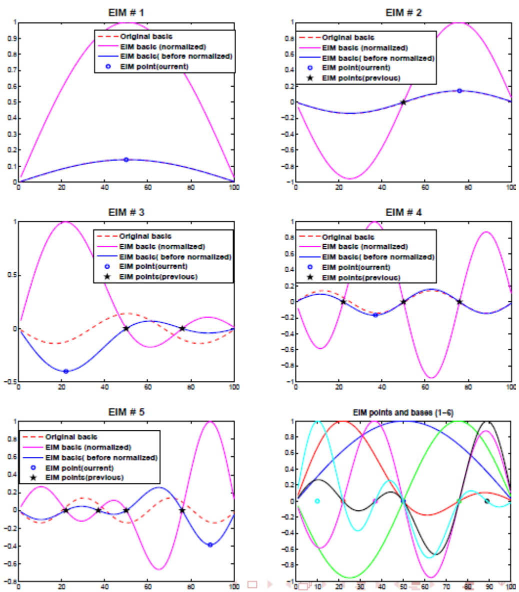


Fig.2: construction of the EIM base (source: private communication from S.Chaturantabut)

Algorithm 1 EIM algorithm

Input : matrix $\mathbf{F} = \begin{bmatrix} - & \mathbf{v}_1 & - \\ & \dots & \\ - & \mathbf{v}_n & - \end{bmatrix}$, factors rank r

Initialization :

$$\begin{aligned} \mathbf{v}_{max} &= max_{\|\cdot\|_\infty}(\mathbf{v}_1, ..., \mathbf{v}_n) \\ \rho_1 &= argmax_{\|\cdot\|_\infty}(\mathbf{v}_1, ..., \mathbf{v}_n) \\ \mathbf{b}_1 &= \frac{\mathbf{v}_{max}}{\|\mathbf{v}_{max}\|_\infty} \\ \mathbf{P}_\rho &= [\mathbf{e}_{\rho_1}] \\ \mathbf{B} &= \begin{bmatrix} - & b_1 & - \end{bmatrix} \end{aligned}$$

Iteration :

for $i = 1, ..., r$ **do**

$$\begin{aligned} \mathbf{C} &= \mathbf{F}\mathbf{P}_\rho(\mathbf{B}\mathbf{P}_\rho)^{-1} \\ \mathbf{R} &= \begin{bmatrix} - & \mathbf{r}_1 & - \\ & \dots & \\ - & \mathbf{r}_n & - \end{bmatrix} = \mathbf{F} - \mathbf{C}\mathbf{B} \\ \mathbf{v}_{max} &= max_{\|\cdot\|_\infty}(\mathbf{r}_1, ..., \mathbf{r}_n) \\ \rho_i &= argmax_{\|\cdot\|_\infty}(\mathbf{r}_1, ..., \mathbf{r}_n) \\ \mathbf{b}_i &= \frac{\mathbf{v}_{max}}{\|\mathbf{v}_{max}\|_\infty} \\ \mathbf{P}_\rho &= [\mathbf{e}_{\rho_1}, ..., \mathbf{e}_{\rho_i}] \\ \mathbf{B} &= \begin{bmatrix} - & \mathbf{b}_1 & - \\ & \dots & \\ - & \mathbf{b}_i & - \end{bmatrix} \end{aligned}$$

end for

Output : basis $\mathbf{B}$, magic points $\mathbf{P}_\rho$

Disgression : Least Squares Extrapolation

- Data extrapolation relies on the fact that **EIM only uses a subset of the data matrix columns for dimension reduction**

- Same could be applied to Least Squares regression if we could find relevant column indices to sample at

- Methods to perform such sampling have been developed recently: see Cohen and Migliorati, *Optimal weighted least-squares methods.*

- Pitfall : here, the number of sampling points must be greater than the rank (typically, $3r$ points are needed for stability)

- I started comparing both methods… The game is close !

Big Data Framework

Description of the data

- Use of a widely-used benchmark to operate on representative data. Set of 12 fuel assemblies sufficient to perform a core calculation.
- Resulting matrix : 1.2 million rows, 4704 columns → 60 Go in HDF5 format. Doesn't easily fit in RAM…

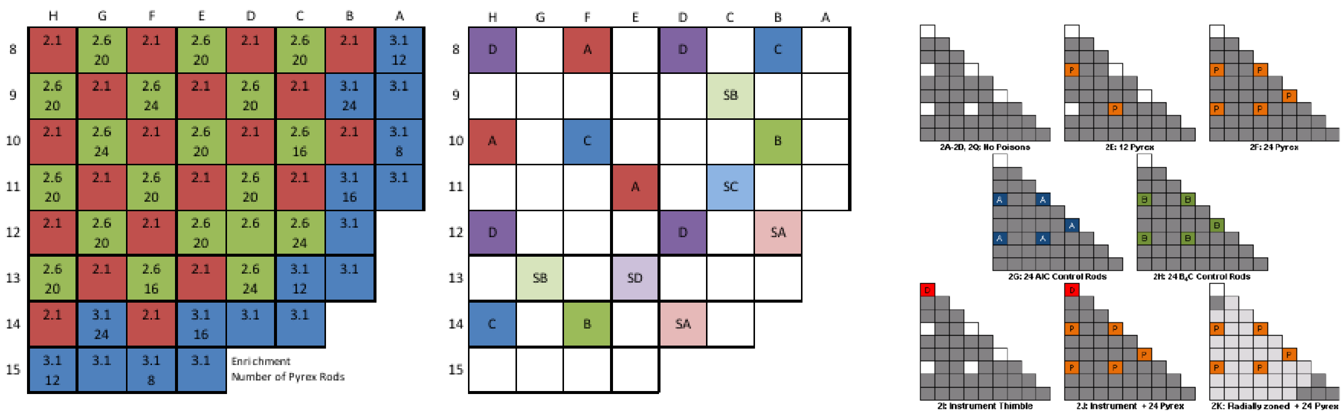
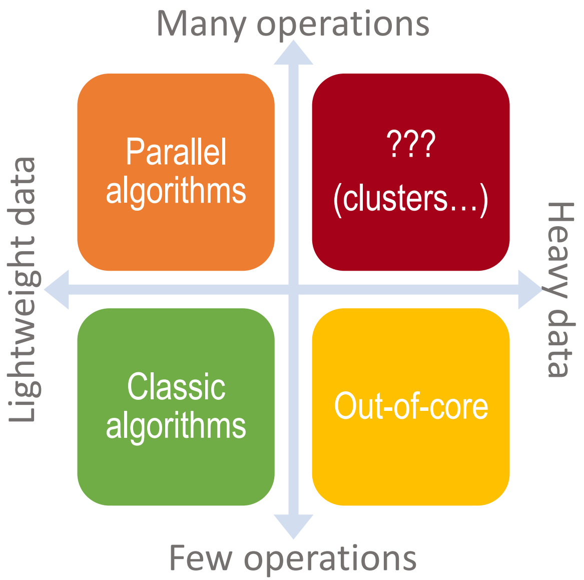


Fig.3: Description of the VERA benchmark (problem n°5)

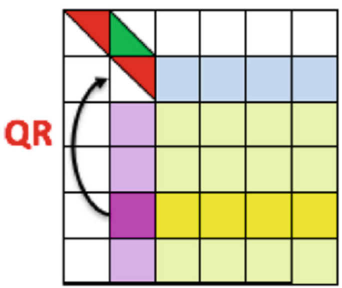
Parallelism and out-of-core computation

- Out-of-core = computing environment where the allocated RAM space is too small to perform the calculation
- In this case, one must:
  - Read the data directly from slow bulk memory;
  - Minimize the number of passes on the data;
  - Minimize the memory footprint (no large volumes of intermediate data);
  - Allocate RAM intelligently
- Counter-intuitively, parallelism is of little use in this case, since performance is capped by the slow read (I/O) anyway

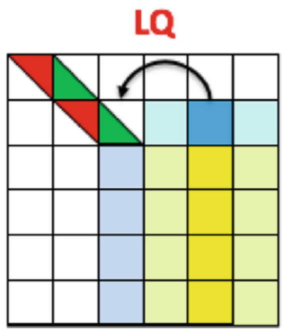


COMPRESSION AND EXTRAPOLATION OF HOMOGENIZED CROSS-SECTIONS BY THE EIM METHOD

- Algorithms exist for out-of-core SVD, but they are recent, complex and not well implemented in accessible libraries
- **EIM adapts readily to out-of-core and parallel computation** : the computation of residues (central step of the database construction) is done independently line by line, and its output is a single value that is easy to store
- A Python library for the management of very large data: the `Dask` library (dynamic task scheduling)



(a) QR factorization of tile $A_{2,2}$

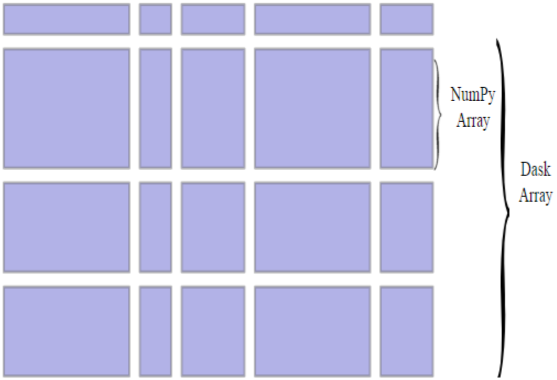


(b) LQ factorization of tile $A_{2,3}$

Fig.5: Representation of an exact out-of-core SVD algorithm (source : K.Kabir & al, *A Framework for Out of Memory SVD Algorithms)*

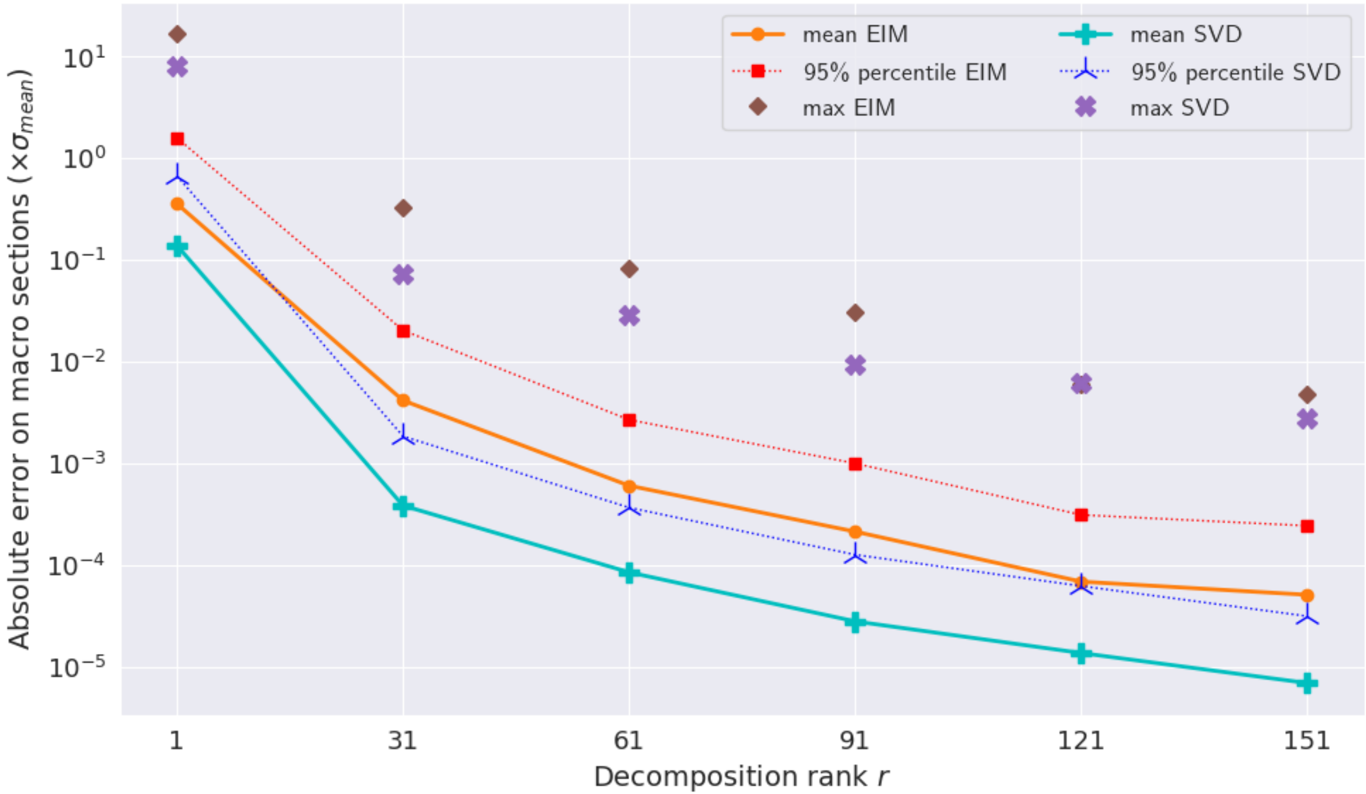
Stochastic EIM

- Even if compression time is not a decisive criterion of performance (offline phase), compressing the whole dataset can be long (1-2h)
- Cause : EIM processes the whole dataset at each iteration, even though it is extremely redundant
- One could rather choose to read only a subset of rows at each iteration and pick the future basis function among these. If the data is already chunked, these subsets can be horizontal groups of chunks
- Very efficient scheme (tested in the next part) !



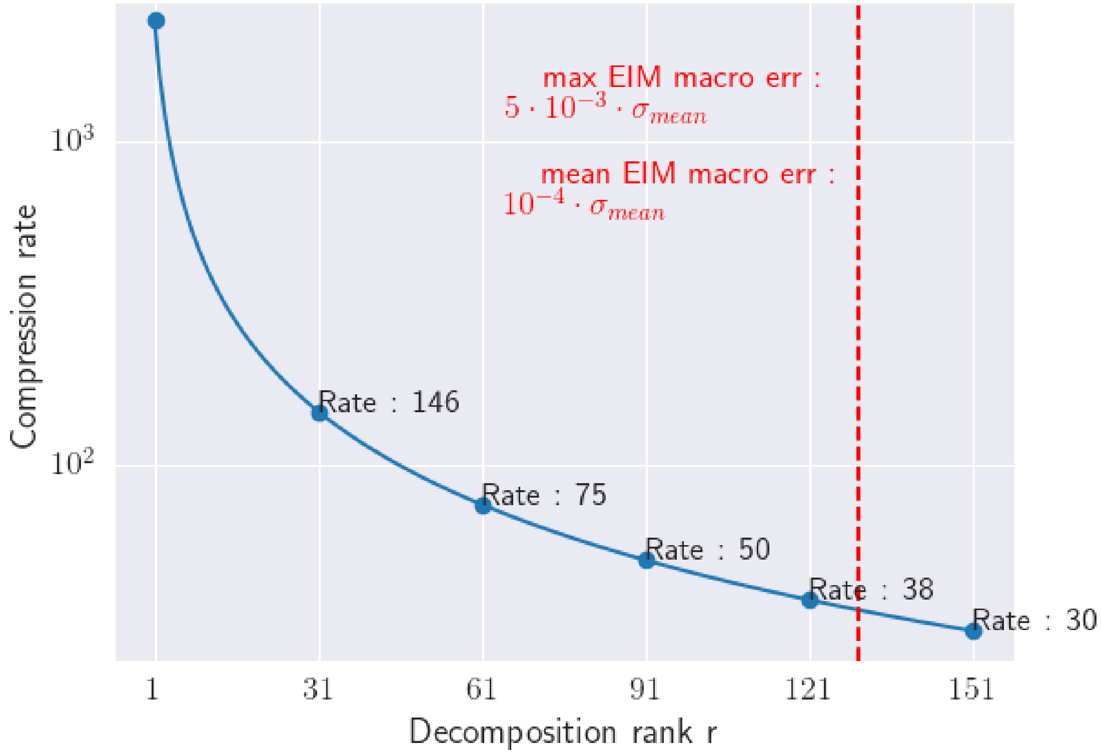
Numerical experiments

COMPRESSION AND EXTRAPOLATION OF HOMOGENIZED CROSS-SECTIONS BY THE EIM METHOD



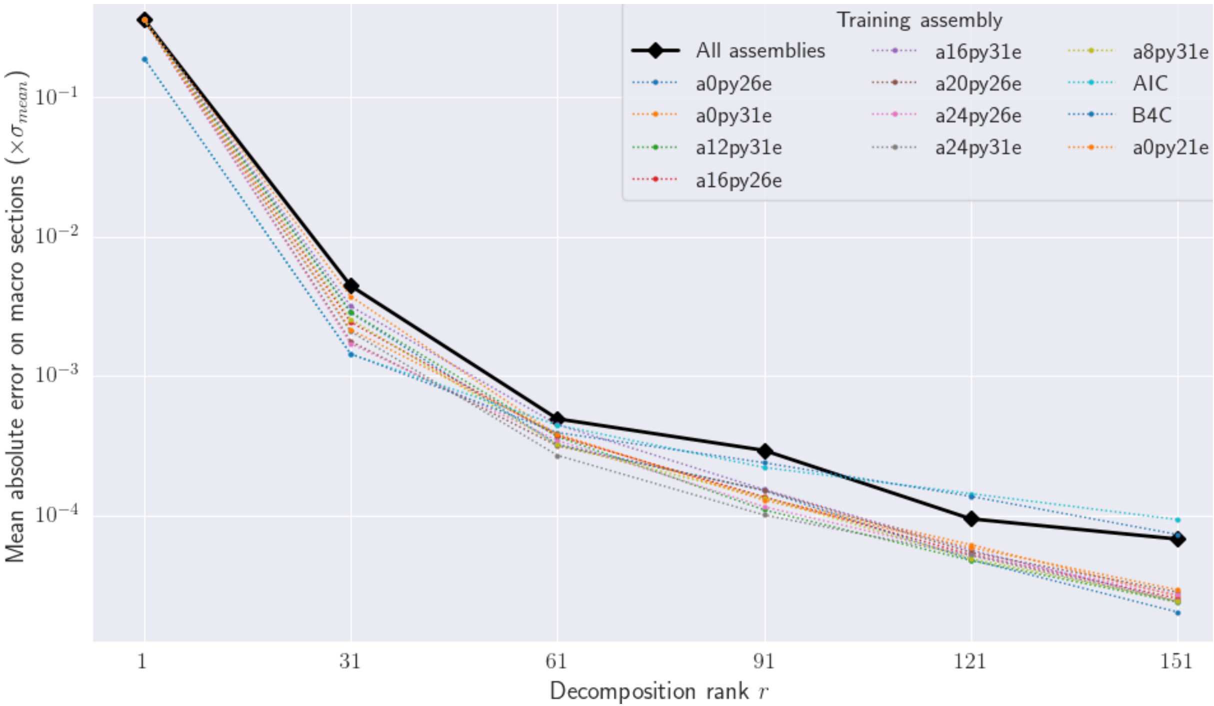
- For compression only, EIM worse than SVD by a small factor (<10)
- Polynomial decay of the error on this data

COMPRESSION AND EXTRAPOLATION OF HOMOGENIZED CROSS-SECTIONS BY THE EIM METHOD



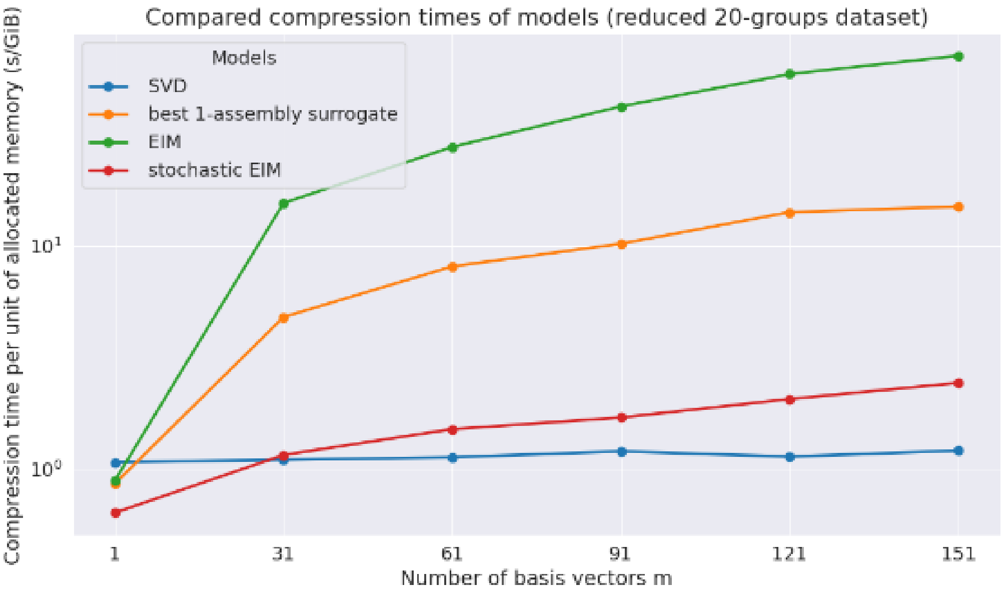
- Compression rates of several dozens achieved.
- The limiting factor is the small side of the matrix → use of tensor methods ?

COMPRESSION AND EXTRAPOLATION OF HOMOGENIZED CROSS-SECTIONS BY THE EIM METHOD



- Impressive extrapolation performance : for most metrics, extrapolated data is more accurate than compressed data !
- Robust to the choice of the training set in our case (except for pathological cases)
- With extrapolation, **88% of overall physics computation was spared**

Compression time performance



- Classical EIM is penalized by its numerous passes on the data
- Stochastic EIM is 40 times faster, and just as accurate
- Surrogate EIM is faster because it works on less data

Convergence exponents and stopping criterion

Performance recap

	r=150, $\mathcal{R}$=30			
Model	Err_{mean}^{rel} $(\cdot\, 10^{-5})$	Err_{max}^{abs} $(\cdot\, \mathbf{10^{-5}}\boldsymbol{\sigma_{moy}})$	T_{comp} (s/GiB)	T_{tot} (s)
SVD	2	284	1	9E4
EIM	10	481	71	9E4
Stochastic EIM	8	439	1.1	9E4
Surrogate	8	247	17	1E4

Conclusion et ouvertures

Conclusion

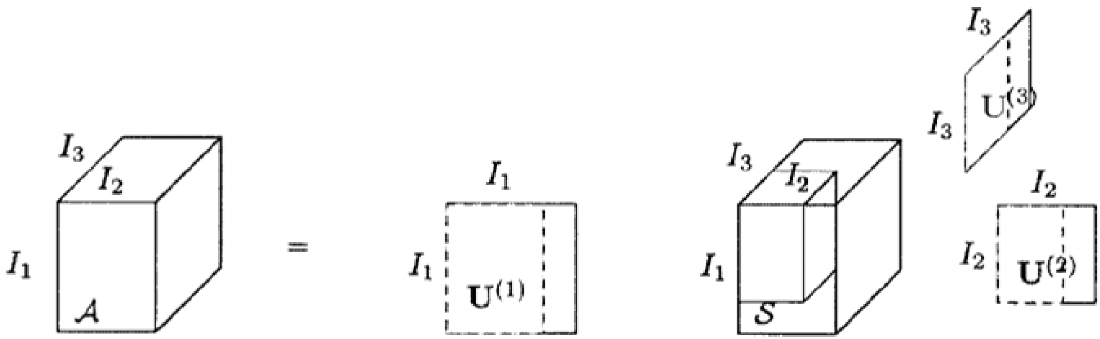
- New use of EIM for compression and extrapolation of physical data
- Philosophy : extensive parametrization can be retrieved from a few parameters values only, by leveraging *self-correlation* of the data
- Simple, linear method, easy to implement even in out-of-core or massively parallel contexts
- Linear decompression commutative with some post-processing operations (slicing, linear combinations, interpolation), allowing for very efficient routines
- **Experimental results on a challenging dataset (60 GB) : reduction of the number of code calls by a factor of 8 and memory savings of a factor 30 – 50 at a negligible accuracy cost**

Looking for new application cases !

- Other physical problems on which to apply this methodology ?
- Desired characteristics :
    - Costly simulation to run for a large number of parameters values (not necessarily a grid)
    - One parameter (continuous or discrete) with many values can be isolated from the other. This is the one that will be extrapolated
    - It’s okay for the resulting snapshots to be correlated
    - Ideally, compression of the data is desired

Development: HOOI and its EIM adaptation

- Compressing the data along several distinct axes (data in tensor form) can improve performance even more
- Optimal algorithm for this : Higher Order Orthogonal Iteration. Same problems as SVD for out-of-core. Slower decompression
- First encouraging results: compression rate x5 with the same average precision!
- Adaptable to the EIM ; creation of a HOEIM? Convergence not guaranteed... Use of the aforementioned least-squares sampling method ?



MERCI POUR VOTRE ATTENTION !

COMPRESSION AND EXTRAPOLATION OF HOMOGENIZED CROSS-SECTIONS BY THE EIM METHOD