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

• Introduction: cross-sections and their reconstruction
• The Empirical Interpolation Method (EIM)
• The Big Data framework
• Experimental results
• Conclusion and perspectives
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 (strongly) correlated multivariate surrogate models, one for each cross-section.

- Double constraint: very high precision (~$10^{-3}$ 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 address 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 transferring 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
  \[ \hat{f}(\tilde{x}, \mu) = \sum_{i=1}^{r} c(\mu) B(\tilde{x}) \approx f(\tilde{x}, \mu) \]

  \[ \Rightarrow \text{Matrix form: } F \approx C \cdot B \text{ (standard matrix decomposition...)} \]

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

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

  \[ \Rightarrow \text{Matrix form: } C = FP_\rho \left( BP_\rho \right)^{-1}, \text{ with } P_\rho \text{ projection matrix on the interpolation points} \]
Application to matrix compression

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

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

→ 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 \( f \in \mathbb{R}^p \) be a matrix row, \( \tilde{f} \in \mathbb{R}^r \) its values at the points \( \rho_1, \ldots, \rho_r \). We can interpolate the compressed coefficients of \( f \) by \( c = \tilde{f}(BP_\rho)^{-1} \), then reconstruct the full vector \( f : f \approx c \cdot B \).

- In other words: if we already have a base and interpolation points, we can generate the code outputs at the points \( \rho_i \) only, and interpolate all the remaining values.

\[ /!\ backslash \text{This “interpolation” is discrete, only aimed to spare some computation:} /!\ backslash \text{we cannot say anything outside of the predefined support points!} \]
Application to extrapolation

Columns picked by the algorithm

Data used for basis + interpolation points construction

Data generated for extrapolation

Extrapolated values

Matrix $F$
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)](image)

**Algorithm 1 EIM algorithm**

Input: matrix $F = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$, factors rank $r$

**Initialization:**

$$v_{\text{max}} = \max_{i} || v_1, \ldots, v_n ||_{\infty}$$

$$\rho_1 = \arg\max_{i} || (v_1, \ldots, v_n) ||_{\infty}$$

$$b_1 = \frac{v_{\text{max}}}{|| v_{\text{max}} ||_{\infty}}$$

$$P_{\rho} = [e_{\rho_1}]$$

$B = \begin{bmatrix} - & b_1 & - \\ \vdots & \vdots & \vdots \\ - & b_r & - \end{bmatrix}$

**Iteration:**

for $i = 1, \ldots, r$

$$C = FP_{\rho}(BP_{\rho})^{-1}$$

$$R = \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} = F - CB$$

$$v_{\text{max}} = \max_{i} || r_1, \ldots, r_n ||_{\infty}$$

$$\rho_i = \arg\max_{i} || (r_1, \ldots, r_n) ||_{\infty}$$

$$b_i = \frac{v_{\text{max}}}{|| v_{\text{max}} ||_{\infty}}$$

$$P_{\rho} = [e_{\rho_1}, \ldots, e_{\rho_n}]$$

$B = \begin{bmatrix} - & b_1 & - \\ \vdots & \vdots & \vdots \\ - & b_r & - \end{bmatrix}$

end for

Output: basis $B$, magic points $P_{\rho}$
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
- 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).

---

**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
• For compression only, EIM worse than SVD by a small factor (<10)
• Polynomial decay of the error on this data
• Compression rates of several dozens achieved.
• The limiting factor is the small side of the matrix → use of tensor methods?
• 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

![Graph showing compared accuracy of models (reduced 20-groups dataset)]

- **Models**
  - SVD
  - Best 1-assembly surrogate
  - EIM

- **Axes**
  - Mean relative error on macro sections (pcm)
  - Compression rate

- **Lines and slopes**
  - SVD: slope = 2.74
  - Best 1-assembly surrogate: slope = 2.65
  - EIM: slope = 2.45
## Performance recap

<table>
<thead>
<tr>
<th>Model</th>
<th>$Err_{\text{rel}}^{\text{mean}}$ ($\cdot 10^{-5}$)</th>
<th>$Err_{\text{abs}}^{\text{max}}$ ($\cdot 10^{-5}\sigma_{\text{moy}}$)</th>
<th>$T_{\text{comp}}$ (s/GiB)</th>
<th>$T_{\text{tot}}$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD</td>
<td>2</td>
<td>284</td>
<td>1</td>
<td>9E4</td>
</tr>
<tr>
<td>EIM</td>
<td>10</td>
<td>481</td>
<td>71</td>
<td>9E4</td>
</tr>
<tr>
<td>Stochastic EIM</td>
<td>8</td>
<td>439</td>
<td>1.1</td>
<td>9E4</td>
</tr>
<tr>
<td>Surrogate</td>
<td>8</td>
<td>247</td>
<td>17</td>
<td>1E4</td>
</tr>
</tbody>
</table>

$r=150$, $R=30$
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