

Randomization in computational linear algebra

Gunnar Martinsson

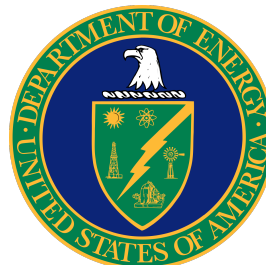
Dept. of Mathematics & Oden Institute for Computational Sciences and Engineering

University of Texas at Austin

Collaborators: Robert van de Geijn, Abinand Gopal, Francisco Igual, Yuji Nakatsukasa, Gregorio Quintana-Ortí, Vladimir Rokhlin, Joel Tropp, Mark Tygert, Heather Wilber.

Current and recent students/postdocs: Joar Bagge, Yunhui Cai, Chao Chen, Ke Chen, Simon Dirckx, Yijun Dong, Michael Han, Nathan Heavner, Joseph Kump, James Levitt, Kate Pearce, Anna Yesypenko.

Research support by:



OUTLINE

- **High accuracy methods for linear algebraic problems**

Main topic.

- Low rank approximation: The randomized SVD. *Key success story.*
- Streaming and single-pass methods.
- Structured random maps (“fast Johnson-Lindenstrauss transforms”).
- Linear solvers: Sketch-to-solve vs. Sketch-to-precondition.
- Las Vegas style methods: Extremely robust and reliable.

- **Sampling based methods for huge problems**

Very brief!

- Monte Carlo style methods → less reliable, less accurate, less robust.
- Enable the solution of stupendously large problems that would otherwise be intractable.

- **Randomized methods for approximating continuum operators**

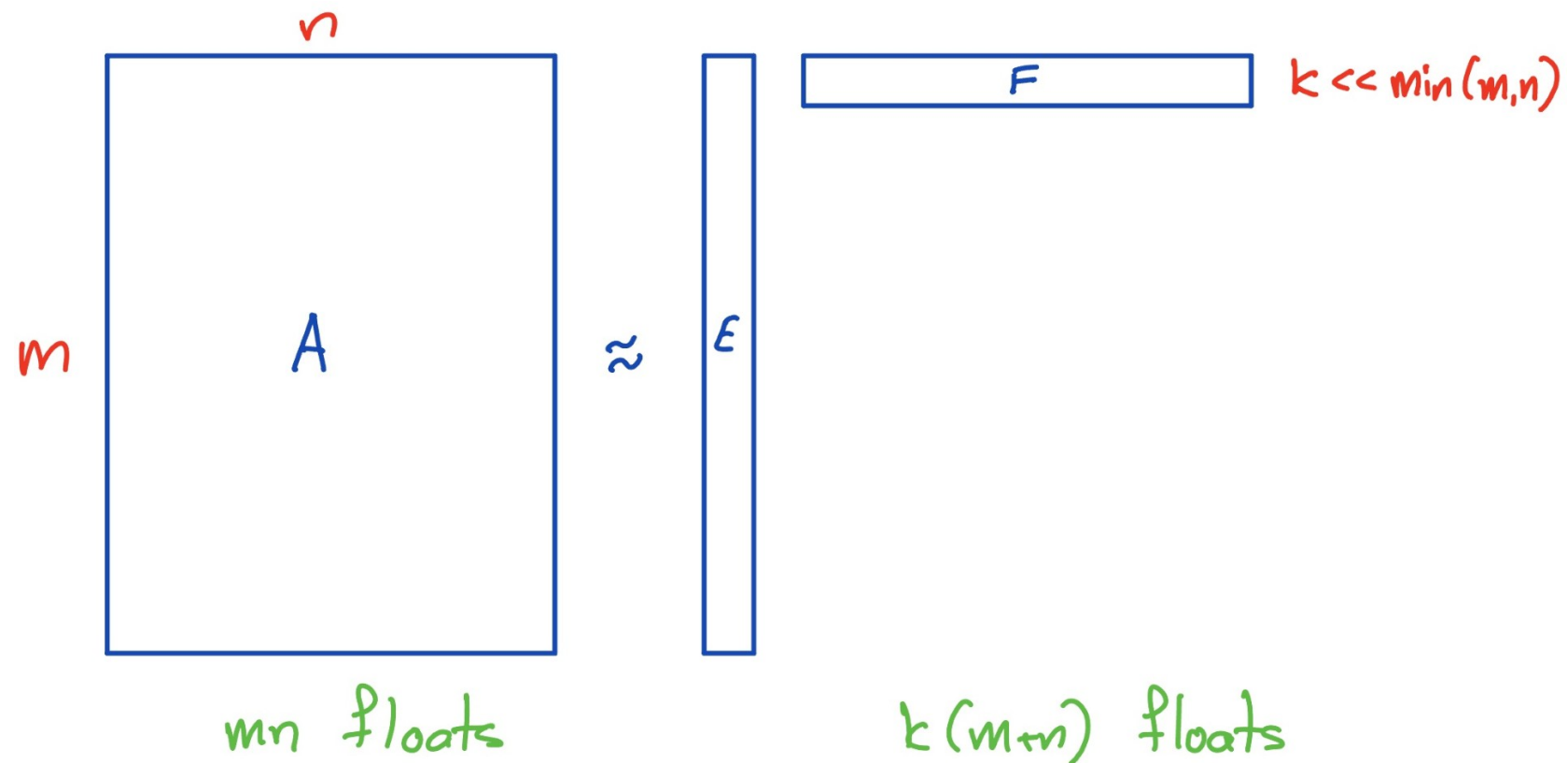
- Approximation of global operators of mathematical physics (solution operators, DtNs, ...).
- Tools for matrices that are not of global low rank, but have structure that can be exploited.
- Conceptually related to Fast Multipole Methods, Fast Direct Solvers, Calderón-Zygmund theory ...

Low rank approximation — problem formulation:

Let \mathbf{A} be a given $m \times n$ matrix, and let k be an integer such that $1 \leq k \ll \min(m, n)$.

We seek to compute approximate factors \mathbf{E} and \mathbf{F} such that

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{E} & \mathbf{F}^* & \\ m \times n & & m \times k & k \times n & \end{array}$$



An example of data compression.

Low rank approximation — problem formulation:

Let \mathbf{A} be a given $m \times n$ matrix, and let k be an integer such that $1 \leq k \ll \min(m, n)$.

We seek to compute approximate factors \mathbf{E} and \mathbf{F} such that

$$\begin{array}{ccc} \mathbf{A} & \approx & \mathbf{E} \mathbf{F}^* \\ m \times n & & m \times k \quad k \times n \end{array}$$

Why?

- Fitting a hyperplane to a given set of points. Or fitting a multivariate normal distribution to measurements (“principal component analysis”).
- Model reduction in scientific computing.
- Spectral algorithms in data analysis.
- “Fast” algorithms of various types: Fast Multipole Methods, generalizations of the Fast Fourier Transform, fast direct solvers, etc.
- Many, many, many more.

Note: We seek only to control the residual error $\|\mathbf{A} - \mathbf{E}\mathbf{F}^*\|$.

Low rank approximation — problem formulation:

Let \mathbf{A} be a given $m \times n$ matrix, and let k be an integer such that $1 \leq k \ll \min(m, n)$.

We seek to compute approximate factors \mathbf{E} and \mathbf{F} such that

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{E} & \mathbf{F}^* & \\ m \times n & & m \times k & k \times n & \end{array}$$

Why?

- Fitting a hyperplane to a given set of points. Or fitting a multivariate normal distribution to measurements (“principal component analysis”).
- Model reduction in scientific computing.
- Spectral algorithms in data analysis.
- “Fast” algorithms of various types: Fast Multipole Methods, generalizations of the Fast Fourier Transform, fast direct solvers, etc.
- Many, many, many more.

Note: We seek only to control the residual error $\|\mathbf{A} - \mathbf{EF}^*\|$.

Precise problem formulation: Given \mathbf{A} (and possibly k), find an approximate orthonormal basis $\{\mathbf{q}_j\}_{j=1}^k$ for the range of \mathbf{A} , so that $\mathbf{A} \approx \mathbf{QQ}^* \mathbf{A}$, where $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_k]$.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \ \cdots \ \mathbf{q}_k]$. The rank k may or may not be known.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \ \cdots \ \mathbf{q}_k]$. The rank k may or may not be known.

If you have \mathbf{Q} , many “low-rank approximation” tasks become easy.

Example: Suppose \mathbf{A} is symmetric, and you seek an approximate eigenvalue decomposition (EVD). Form the matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}\mathbf{Q}$, and compute the EVD of \mathbf{B} :

$$\mathbf{B} = \mathbf{V}\mathbf{D}\mathbf{V}^*.$$

Then

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{Q}\mathbf{Q}^* = \mathbf{Q}\mathbf{B}\mathbf{Q}^* = \mathbf{Q}\mathbf{V}\mathbf{D}\mathbf{V}^*\mathbf{Q}^* = \{\text{Set } \mathbf{U} := \mathbf{Q}\mathbf{V}\} = \mathbf{U}\mathbf{D}\mathbf{U}^*$$

is an approximate EVD of \mathbf{A} .

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \ \cdots \ \mathbf{q}_k]$. The rank k may or may not be known.

Excellent algorithms for computing \mathbf{Q} already exist:

- **Compute the full SVD and truncate**

- Eckart-Young theorem tells us this is optimal.
- Algorithms are stable, accurate, and reliable. Routinely yield double precision accuracy.
- Algorithms must be iterative, but converge *extremely* fast in practice.
- Cost scales poorly with matrix size, $O(n^3)$ for $n \times n$ matrix. Hard to parallelize.

- **Krylov methods**

For instance, $\text{ran}(\mathbf{Q}) = \text{ran}([\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \dots, \mathbf{A}^{k-1}\mathbf{b}])$.

- Particularly efficient when fast matrix-vector multiplication is available. (E.g. \mathbf{A} is sparse.)
- If the metric is number of matvecs, then Krylov methods are often “optimal”.
- Well understood – precise theory is available in many (but far from all!) cases.

- **Gram-Schmidt (with “column pivoting”)**

- Very simple!
- If column pivoting is used, it is robust and typically works well. (Some intricacies arise.)
- Can be stopped after k steps if required tolerance is met. \rightarrow Complexity $O(mnk)$.
- Communication intensive, so somewhat slow in practice. Does not preserve sparsity.
- Output is quite far from optimal when singular values decay slowly.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

Let us consider Gram-Schmidt in some detail, applied to $\mathbf{A} = [\mathbf{a}_1 \mathbf{a}_2 \cdots \mathbf{a}_n]$.

In the “generic” case, Gram-Schmidt proceeds as follows:

$$\begin{array}{lll}
 \mathbf{a}'_1 = \mathbf{a}_1 & \mathbf{q}_1 = \frac{1}{\|\mathbf{a}'_1\|} \mathbf{a}'_1 & \mathbf{P}_1 = \mathbf{q}_1 \mathbf{q}_1^* \\
 \mathbf{a}'_2 = (\mathbf{I} - \mathbf{P}_1) \mathbf{a}_2 & \mathbf{q}_2 = \frac{1}{\|\mathbf{a}'_2\|} \mathbf{a}'_2 & \mathbf{P}_2 = \mathbf{P}_1 + \mathbf{q}_2 \mathbf{q}_2^* \\
 \mathbf{a}'_3 = (\mathbf{I} - \mathbf{P}_2) \mathbf{a}_3 & \mathbf{q}_3 = \frac{1}{\|\mathbf{a}'_3\|} \mathbf{a}'_3 & \mathbf{P}_3 = \mathbf{P}_2 + \mathbf{q}_3 \mathbf{q}_3^* \\
 \vdots & \vdots & \vdots
 \end{array}$$

The algorithm can fail outright, since any vector \mathbf{a}'_j may be zero.

It can also fail numerically, if \mathbf{a}'_j is small. (Fixable, by swapping to *unitary* transforms.)

Classical fix: Column pivoting – apply projections to all columns, and pick the largest remaining one. Greatly increases cost! And ruins sparsity.

Randomization fix: If you randomly mix the columns first, you can forgo pivoting!

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

Gram-Schmidt with randomized mixing:

For $j = 1, 2, \dots, k$, draw Gaussian random vector ω_j , and form sample vectors

$$\mathbf{y}_j = \mathbf{A}\omega_j.$$

Then perform Gram-Schmidt on the vectors $\{\mathbf{y}_j\}_{j=1}^k$ to form the ON set $\{\mathbf{q}_j\}_{j=1}^k$.

Claim: The matrix $\mathbf{Q} = [\mathbf{q}_1 \mathbf{q}_2 \cdots \mathbf{q}_k]$ is often a good answer to our “task”.

A basic analysis of the scheme is easy. Let $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^*$ be the exact SVD of \mathbf{A} . Then

$$\mathbf{y}_j = \mathbf{A}\omega_j = \mathbf{U}\mathbf{D}\mathbf{V}^*\omega_j = \{\text{Set } \psi_j := \mathbf{V}^*\omega_j\} = \mathbf{U}\mathbf{D}\psi_j.$$

Observe that ψ_j also has a Gaussian distribution. We find that

$$\mathbf{y}_j = \mathbf{U}\mathbf{D}\psi_j = \sum_{i=1}^n \psi_{i,j} \sigma_i \mathbf{u}_i. \quad \text{Note: } \psi_{i,j} \in \mathcal{N}(0, 1) \text{ are i.i.d.}$$

Cost of scheme: $O(mk^2)$ flops + the $O(mnk)$ cost of mixing

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \ \cdots \ \mathbf{q}_k]$. The rank k may or may not be known.

Algorithm: (Assuming k is given in advance.)

1. Draw an $n \times k$ Gaussian random matrix Ω .

$\Omega = \text{randn}(n, k)$

2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A}\Omega$.

$\mathbf{Y} = \mathbf{A} * \Omega$

3. Form an $m \times k$ orthonormal matrix \mathbf{Q} such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$.

$[\mathbf{Q}, \sim] = \text{qr}(\mathbf{Y})$

Recall that

$$\mathbf{y}_j = \sum_{i=1}^n (\sigma_i \psi_{i,j}) \mathbf{u}_i, \quad \text{where } \psi_{i,j} \sim \mathcal{N}(0, 1).$$

Quick facts:

- When \mathbf{A} has *exact* rank k , the algorithm succeeds (i.e. $\mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$) with probability 1.
- When the singular values *decay rapidly*, we expect close to optimal results.
- When the singular values *decay slowly*, the basis will likely not be good.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

Algorithm: (Assuming k is given in advance.)

1. Draw an $n \times k$ Gaussian random matrix Ω .

$\Omega = \text{randn}(n, k)$

2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A}\Omega$.

$\mathbf{Y} = \mathbf{A} * \Omega$

3. Form an $m \times k$ orthonormal matrix \mathbf{Q} such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$.

$[\mathbf{Q}, \sim] = \text{qr}(\mathbf{Y})$

Recall that

$$\mathbf{y}_j = \sum_{i=1}^n (\sigma_i \psi_{i,j}) \mathbf{u}_i, \quad \text{where } \psi_{i,j} \sim \mathcal{N}(0, 1).$$

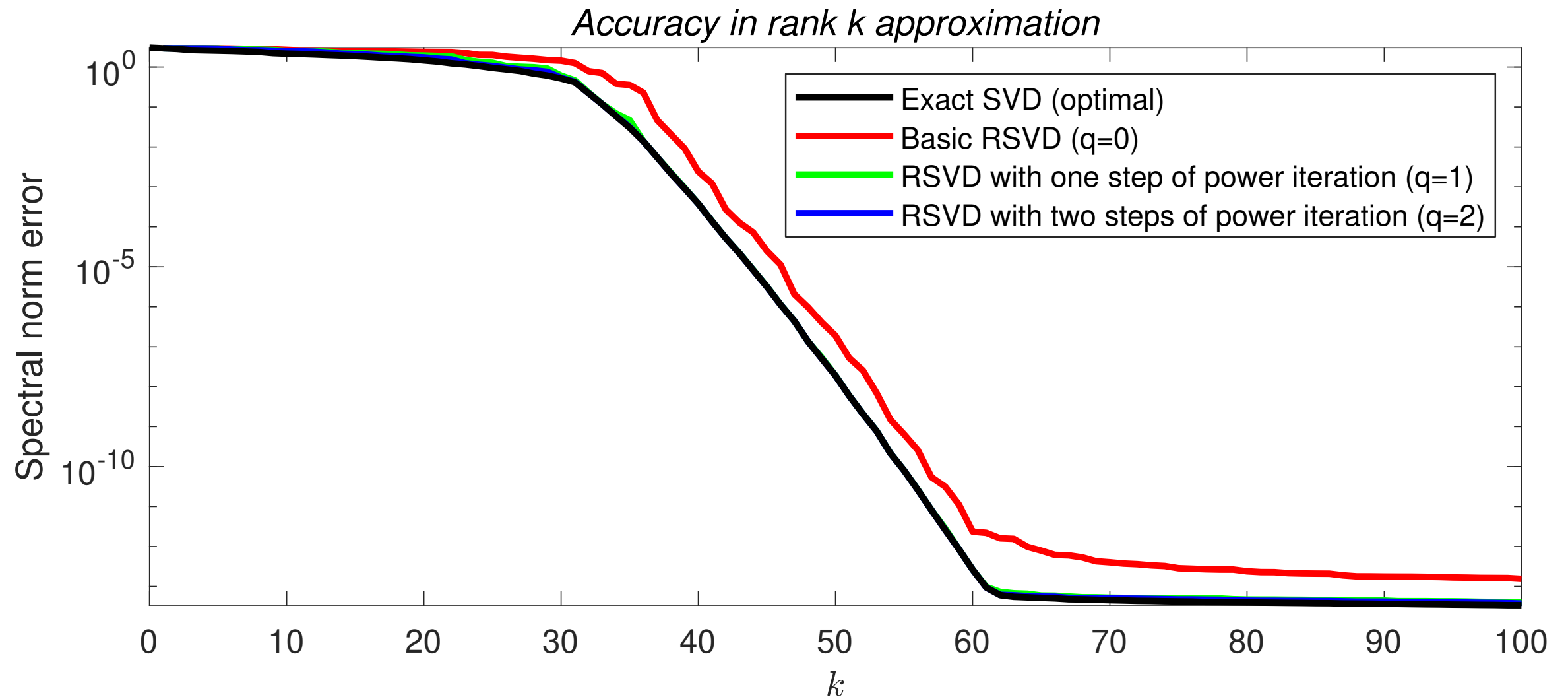
Quick facts:

- When \mathbf{A} has *exact* rank k , the algorithm succeeds (i.e. $\mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$) with probability 1.
- When the singular values *decay rapidly*, we expect close to optimal results.
- When the singular values *decay slowly*, the basis will likely not be good.

This can be fixed via powering, to boost the separation between large/small svds.

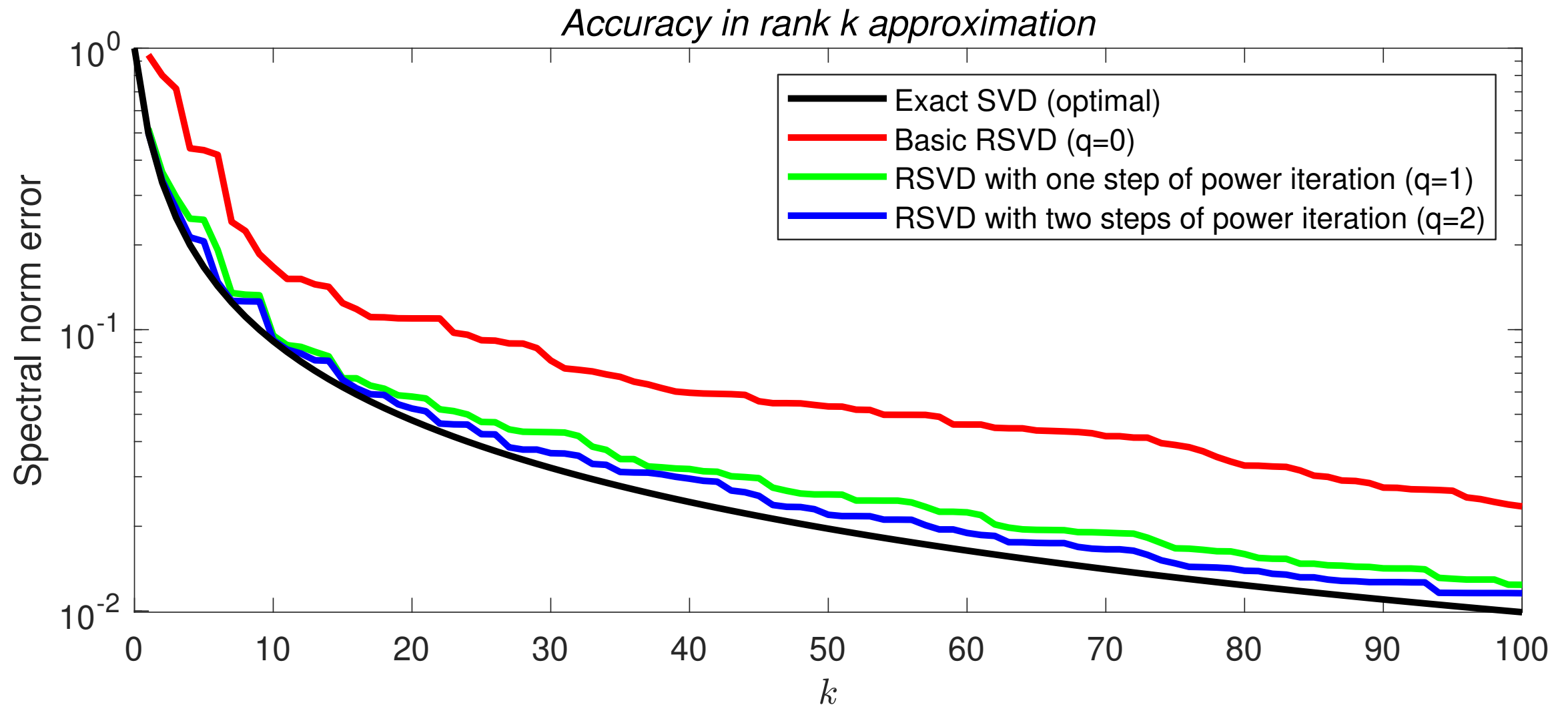
For instance, replace $\mathbf{Y} = \mathbf{A}\Omega$ by $\mathbf{Y} = \mathbf{A}\mathbf{A}^\mathbf{A}\Omega$. Then $\mathbf{y}_j = \sum_{i=1}^n (\sigma_i^3 \psi_{i,j}) \mathbf{u}_i$.*

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.



The plot shows the errors $e_k = \|\mathbf{A} - \mathbf{P}_k \mathbf{A}\|$, where \mathbf{P}_k is the orthogonal proj. onto the span of the first k columns of $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{\Omega}$, and where $\mathbf{\Omega}$ is a Gaussian random matrix. (The matrix \mathbf{A} is an approximation to a scattering operator for a Helmholtz problem.)

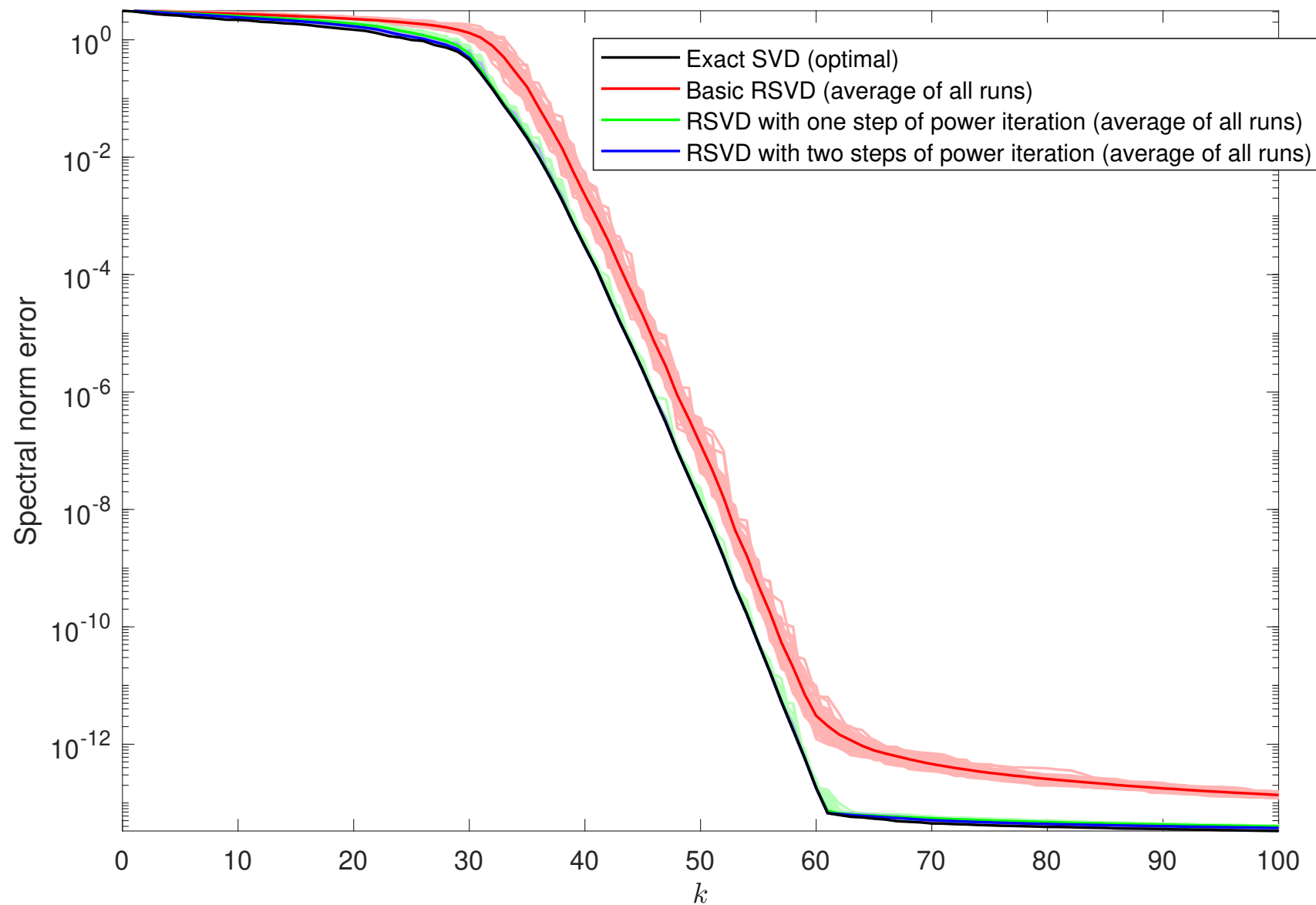
Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.



The plot shows the errors $e_k = \|\mathbf{A} - \mathbf{P}_k \mathbf{A}\|$, where \mathbf{P}_k is the orthogonal proj. onto the span of the first k columns of $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{\Omega}$, and where $\mathbf{\Omega}$ is a Gaussian random matrix. (The matrix \mathbf{A} now has singular values that decay slowly.)

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

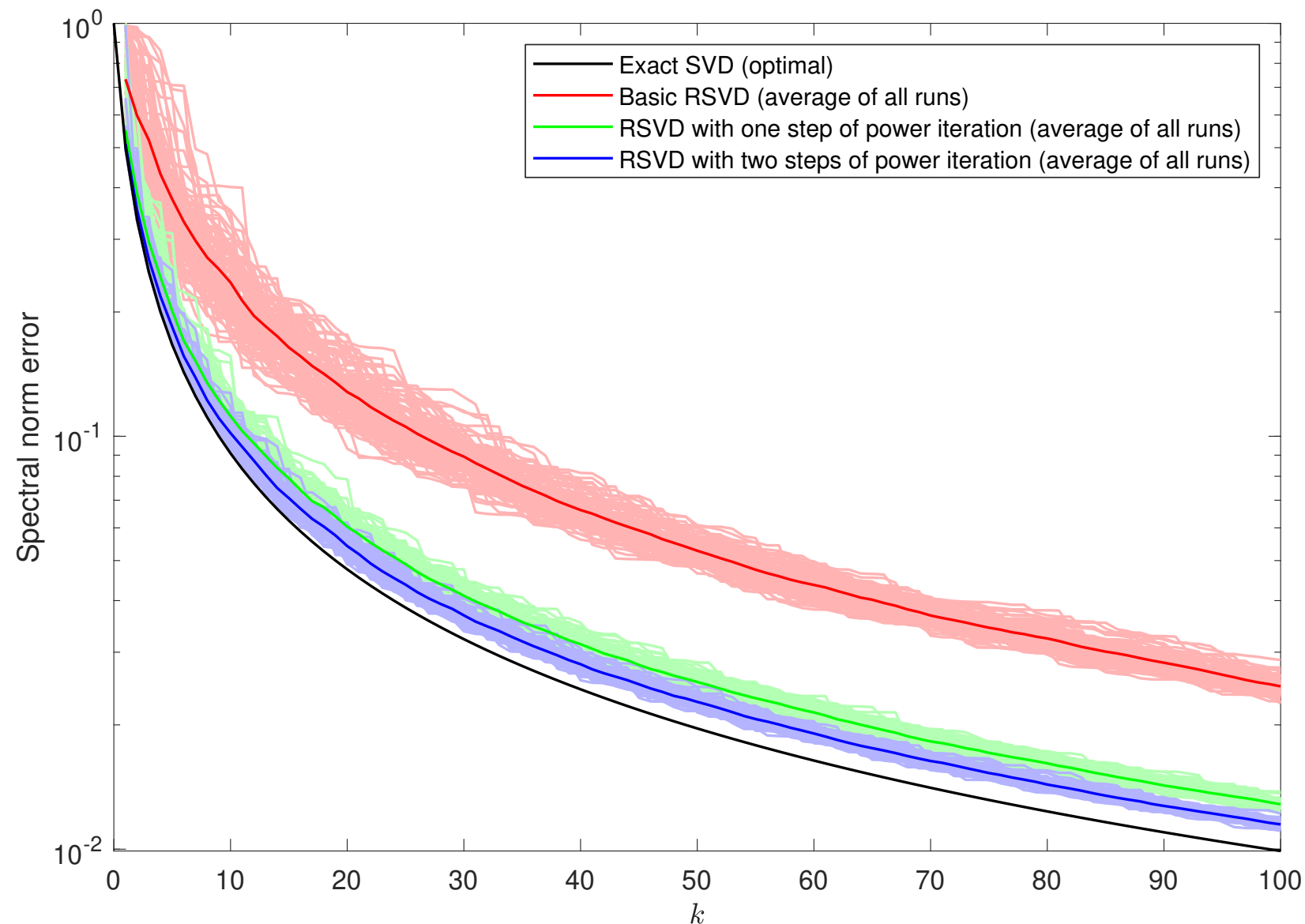
The same plot as before, but now showing 100 instantiations.



The darker lines show the mean errors across the 100 experiments.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \ \cdots \ \mathbf{q}_k]$. The rank k may or may not be known.

The same plot as before, but now showing 100 instantiations.



The darker lines show the mean errors across the 100 experiments.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

The probabilistic behavior of the error has been studied extensively, and reasonably tight theoretical bounds have been established.

A small sample of references addressing this and related problems:

- Frieze, Kannan & Vempala, 2004.
- Martinsson, Rokhlin & Tygert, YALEU/DCS/RR-1361, 2006.
- Liberty, Woolfe, Martinsson, Rokhlin & Tygert, PNAS, 2008.
- Halko, Martinsson & Tropp, *SIAM Review*, 2011.
- Witten & Candès, *Algorithmica*, 2015.
- Gu, *SISC*, 2015. Analysis of randomized subspace iteration.
- Musco & Musco, *NIPS*, 2015. Analysis of block Krylov methods.
- Saibaba, *SIMAX*, 2019. Accuracy of singular vectors.
- Martinsson & Tropp, *Acta Numerica*, 2020.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

To illustrate, suppose that we seek to match the optimal error in a rank- k approximation. We apply the basic randomized procedure involving $\ell \geq k$ samples.

1. Draw an $n \times \ell$ Gaussian random matrix Ω .

`Omega = randn(n,l)`

2. Form the $m \times \ell$ sample matrix $\mathbf{Y} = \mathbf{A}\Omega$.

`Y = A * Omega`

3. Form an $m \times \ell$ orthonormal matrix \mathbf{Q} such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$.

`[Q, ~] = qr(Y)`

The resulting error then satisfies

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\boldsymbol{\Psi}_{(n-k),\ell}\boldsymbol{\Psi}_{k,\ell}^\dagger\|^2$$

where “ $\boldsymbol{\Psi}_{i,j}$ ” denotes an $i \times j$ matrix drawn from a Gaussian distribution, and where

$\mathbf{D}_2 = \text{diag}(\sigma_{k+1}, \sigma_{k+2}, \dots, \sigma_{\min(m,n)})$ holds the “tail” singular values of \mathbf{A} .

Note: The term $\|\mathbf{D}_2\|^2$ is the minimal error, according to Eckart-Young.

The suboptimality term depends on $\sigma_{\min}(\boldsymbol{\Psi}_{k,\ell})$, where $\boldsymbol{\Psi}_{k,\ell}$ is a $k \times \ell$ Gaussian matrix.

- With no oversampling, $k = \ell$, we get terrible results!
- With lots of oversampling, $k \ll \ell$, things look good. (“Marchenko–Pastur” case)

What about the case where ℓ is only slightly larger than k ?

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

To illustrate, suppose that we seek to match the optimal error in a rank- k approximation. We apply the basic randomized procedure involving $\ell \geq k$ samples.

1. Draw an $n \times \ell$ Gaussian random matrix $\mathbf{\Omega}$.

$\text{Omega} = \text{randn}(n, \ell)$

2. Form the $m \times \ell$ sample matrix $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

$\mathbf{Y} = \mathbf{A} * \text{Omega}$

3. Form an $m \times \ell$ orthonormal matrix \mathbf{Q} such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$.

$[\mathbf{Q}, \sim] = \text{qr}(\mathbf{Y})$

The resulting error then satisfies

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\mathbf{\Psi}_{(n-k),\ell}\mathbf{\Psi}_{k,\ell}^\dagger\|^2$$

where “ $\mathbf{\Psi}_{i,j}$ ” denotes an $i \times j$ matrix drawn from a Gaussian distribution, and where

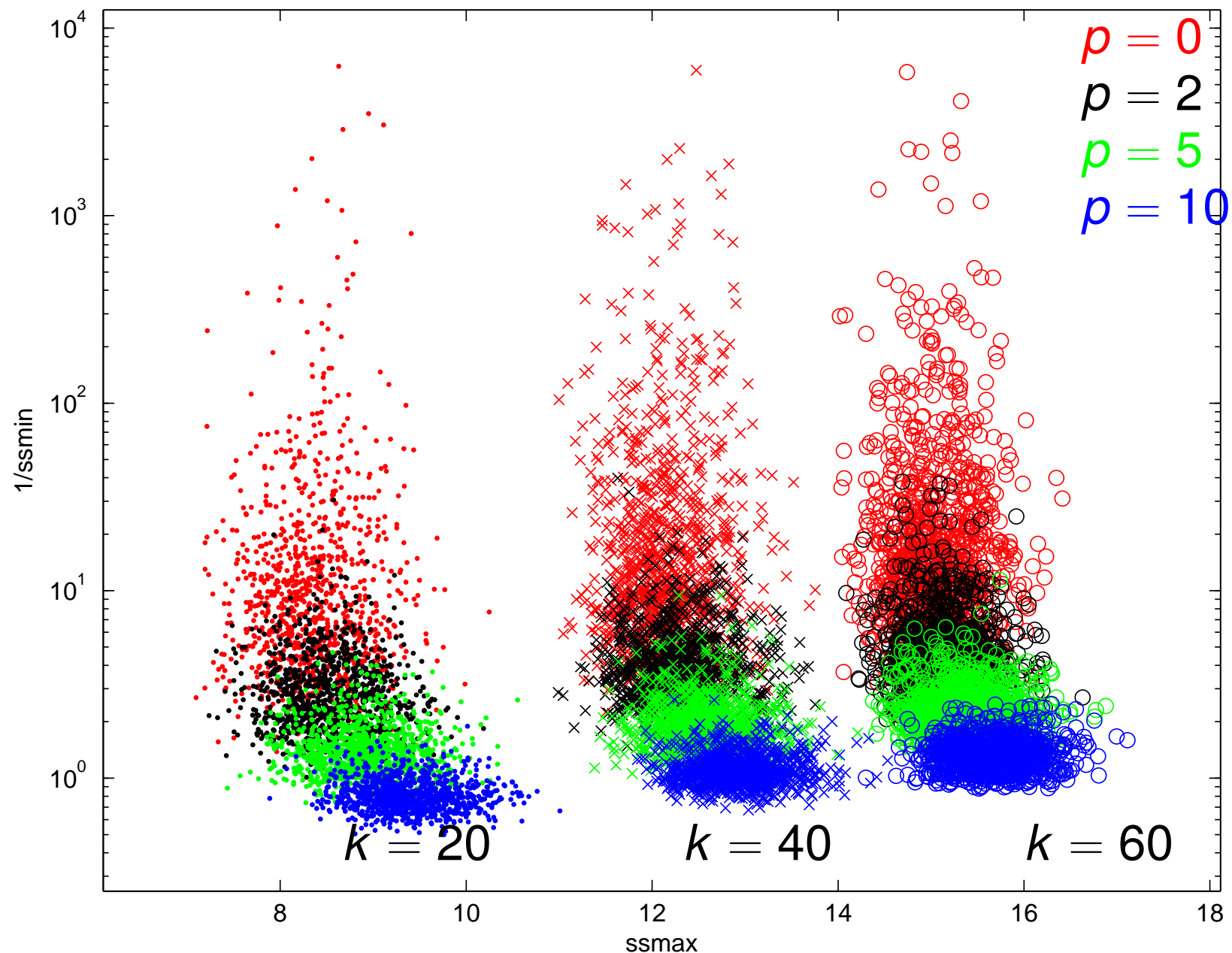
$\mathbf{D}_2 = \text{diag}(\sigma_{k+1}, \sigma_{k+2}, \dots, \sigma_{\min(m,n)})$ holds the “tail” singular values of \mathbf{A} .

Note: The term $\|\mathbf{D}_2\|^2$ is the minimal error, according to Eckart-Young.

Proposition: (Chen & Dongarra 2005) Let $\mathbf{\Psi}$ be a Gaussian matrix of size $k \times k + p$ where $p \geq 2$. Then $(\mathbb{E}[\|\mathbf{\Psi}^\dagger\|_F^2])^{1/2} \leq \sqrt{\frac{k}{p-1}}$ and $\mathbb{E}[\|\mathbf{\Psi}^\dagger\|] \leq \frac{e\sqrt{k+p}}{p}$.

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

Scatter plot showing distribution of $1/\sigma_{\min}$ for $k \times (k + p)$ Gaussian matrices.



$1/\sigma_{\min}$ is plotted against σ_{\max} .

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

Using the bound $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\boldsymbol{\Psi}_{(n-k),\ell}\boldsymbol{\Psi}_{k,\ell}^\dagger\|^2$, along with the bounds on $\sigma_{\min}(\boldsymbol{\Psi}_{k,\ell})$, we can bound the expectation of the error:

Theorem: (Halko, Martinsson, Tropp 2011) Let \mathbf{A} be an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Let k be a target rank, and let p be an over-sampling parameter such that $p \geq 2$ and $k + p \leq \min(m, n)$. Let $\boldsymbol{\Omega}$ be a Gaussian random matrix of size $n \times (k + p)$ and set $\mathbf{Q} = \text{orth}(\mathbf{A}\boldsymbol{\Omega})$. Then the average error satisfies

$$\mathbb{E}[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}}] \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2},$$

$$\mathbb{E}[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|] \leq \left(1 + \sqrt{\frac{k}{p-1}}\right) \sigma_{k+1} + \frac{e\sqrt{k+p}}{p} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}.$$

There are also bounds on the likelihood of a large deviation from the expectation. (It turns out to decay super-exponentially fast as p increases!)

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

Let us now incorporate over-sampling by p vectors (say $p = 5$ or $p = 10$):

1. Draw an $n \times (k + p)$ Gaussian random matrix Ω .

`Omega = randn(n,k+p)`

2. Form the $m \times (k + p)$ sample matrix $\mathbf{Y} = \mathbf{A}\Omega$.

`Y = A * Omega`

3. Form an $m \times (k + p)$ orthonormal matrix \mathbf{Q} such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$.

`[Q, ~] = qr(Y)`

Task: Let \mathbf{A} be an $m \times n$ matrix of approximately low rank. We seek to build an approximate ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for its range. The objective is to make $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ small where $\mathbf{Q} = [\mathbf{q}_1 \cdots \mathbf{q}_k]$. The rank k may or may not be known.

Let us now incorporate over-sampling by p vectors (say $p = 5$ or $p = 10$):

1. Draw an $n \times (k + p)$ Gaussian random matrix Ω . `Omega = randn(n,k+p)`
2. Form the $m \times (k + p)$ sample matrix $\mathbf{Y} = \mathbf{A}\Omega$. `Y = A * Omega`
3. Form an $m \times (k + p)$ orthonormal matrix \mathbf{Q} such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

Important: It turns out to be easy to “filter out” any excess information gathered as a result of over-sampling:

4. Form the $(k + p) \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`
5. Form the full SVD of the small matrix \mathbf{B} : $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B, 'econ')`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

We refer to this full algorithm as the *randomized SVD (RSVD)*.

(Optionally, the factorization can be truncated to drop the last p singular modes.)

Key point: Over-sampling is entirely safe in this context!

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Observation: Step (4) contributes an additional step of power iteration.

As a consequence, the leading singular values are computed more accurately than the previous theorem might indicate.

Additionally, you get extra accuracy in the *right* singular vectors:

Theorem: (Y. Dong, P.G. Martinsson, Y. Nakatsukasa 2023) For a rank- l randomized SVD with the Gaussian embedding and $q \geq 0$ power iterations, when the oversampled rank l satisfies $l = \Omega(k)$ (where k is the target rank, $k < l < r = \text{rank}(\mathbf{A})$) and q is

reasonably small such that $\eta = \frac{(\sum_{j=k+1}^r \sigma_j^{4q+2})^2}{\sum_{j=k+1}^r \sigma_j^{2(4q+2)}}$ satisfies $\eta = \Omega(l)$, with high probability (at least $1 - e^{-\Theta(k)} - e^{-\Theta(l)}$), there exist distortion factors $0 < \epsilon_1, \epsilon_2 < 1$ such that

$$(1) \quad \sin \angle_i (\mathbf{u}_k, \hat{\mathbf{u}}_l) \leq \left(1 + \frac{1 - \epsilon_1}{1 + \epsilon_2} \cdot \frac{l}{\sum_{j=k+1}^r \sigma_j^{4q+2}} \cdot \sigma_i^{4q+2} \right)^{-\frac{1}{2}}$$

$$(2) \quad \sin \angle_i (\mathbf{v}_k, \hat{\mathbf{v}}_l) \leq \left(1 + \frac{1 - \epsilon_1}{1 + \epsilon_2} \cdot \frac{l}{\sum_{j=k+1}^r \sigma_j^{4q+4}} \cdot \sigma_i^{4q+4} \right)^{-\frac{1}{2}}$$

for all $i \in [k]$, where $\epsilon_1 = \Theta\left(\sqrt{\frac{k}{l}}\right)$ and $\epsilon_2 = \Theta\left(\sqrt{\frac{l}{\eta}}\right)$.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Observation: Step (4) contributes an additional step of power iteration.

As a consequence, the leading singular values are computed more accurately than the previous theorem might indicate.

Current work:

A posteriori error estimation — construct bounds that involve only information known to the user at the time of execution.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** Ω .

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\Omega$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Practical note: The algorithm described is very *communication efficient*, as the interaction with \mathbf{A} is through the matrix-matrix multiplication only.

- High practical speed: matrix-matrix multiplication is very parallelizable, and has been highly optimized for many architectures.
- Order of magnitude acceleration for data stored out-of-core.
- Highly efficient for GPU computing, or mobile computing (phones, etc).

Note: Krylov methods share the advantage that the interaction with \mathbf{A} is only via the action of the matrix on vectors. The difference is that in the RSVD, the matrix-vector multiplies happen in batches, which is much more efficient.

Block Krylov methods are more similar in this regard.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (gives correct asymptotic complexity):

Simply try with $k = 1$ first.

Use an a posteriori error estimator to check if the tolerance is met.

If not, then keep doubling, $k = 2, 4, 8, 16, \dots$

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (efficient in practice): Build the factorization iteratively.

For instance, fix a block size b (say $b = 25$), and a requested tolerance ε .

Build a rank- b factorization of \mathbf{A} using the RSVD.

Estimate the error using an a posteriori error estimator.

If the tolerance is not met, then use RSVD again to build a rank- b factorization of *the residual* and add the new basis vectors to the b you got in the first step.

Etc.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

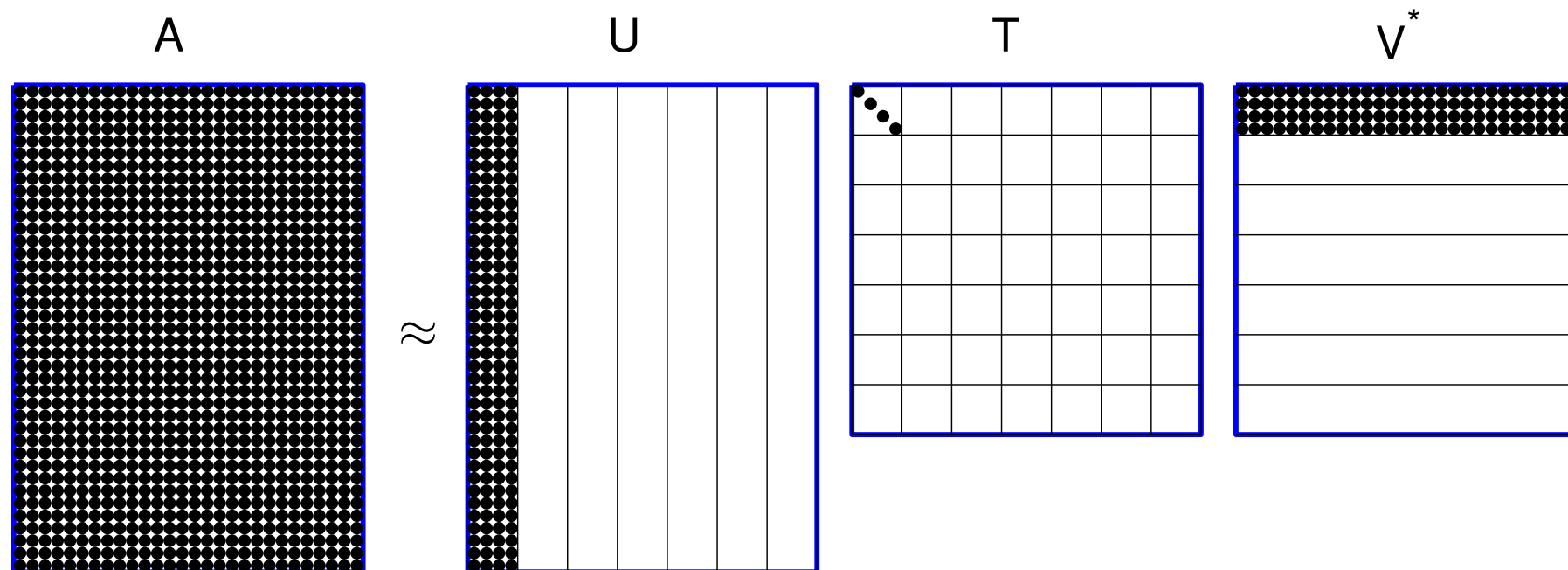
(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (efficient in practice): Build the factorization iteratively.

Illustration for block size $b = 4$:



In the first step, compute a rank 4 approximation using the SVD.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

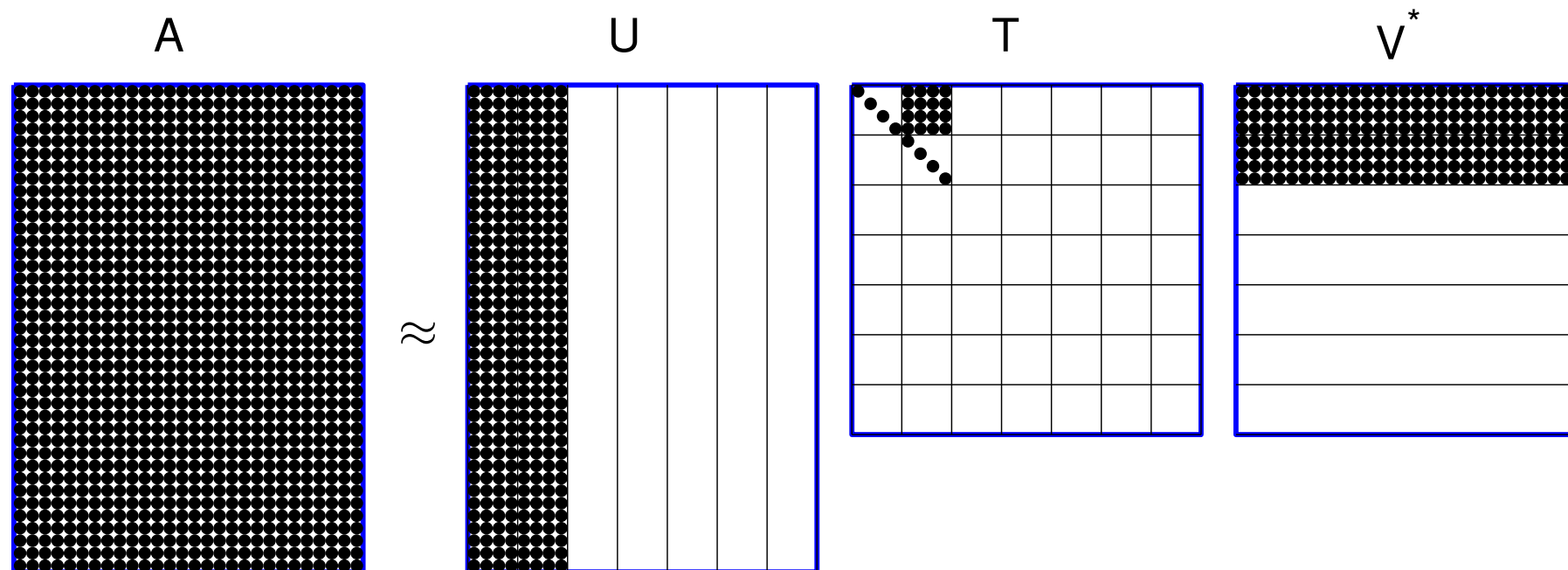
(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (efficient in practice): Build the factorization iteratively.

Illustration for block size $b = 4$:



If the tolerance was not met, do a rank-4 RSVD on the residual to get rank-8 approximation.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

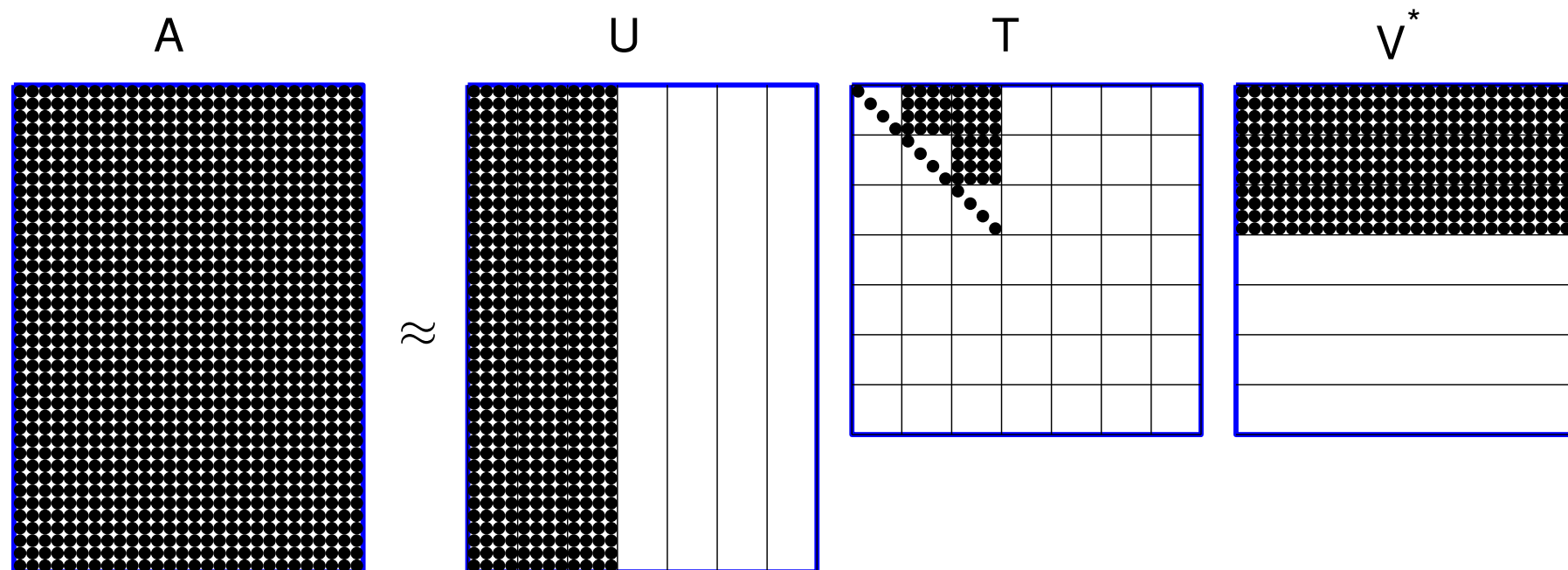
(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (efficient in practice): Build the factorization iteratively.

Illustration for block size $b = 4$:



The rank-12 approximation.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

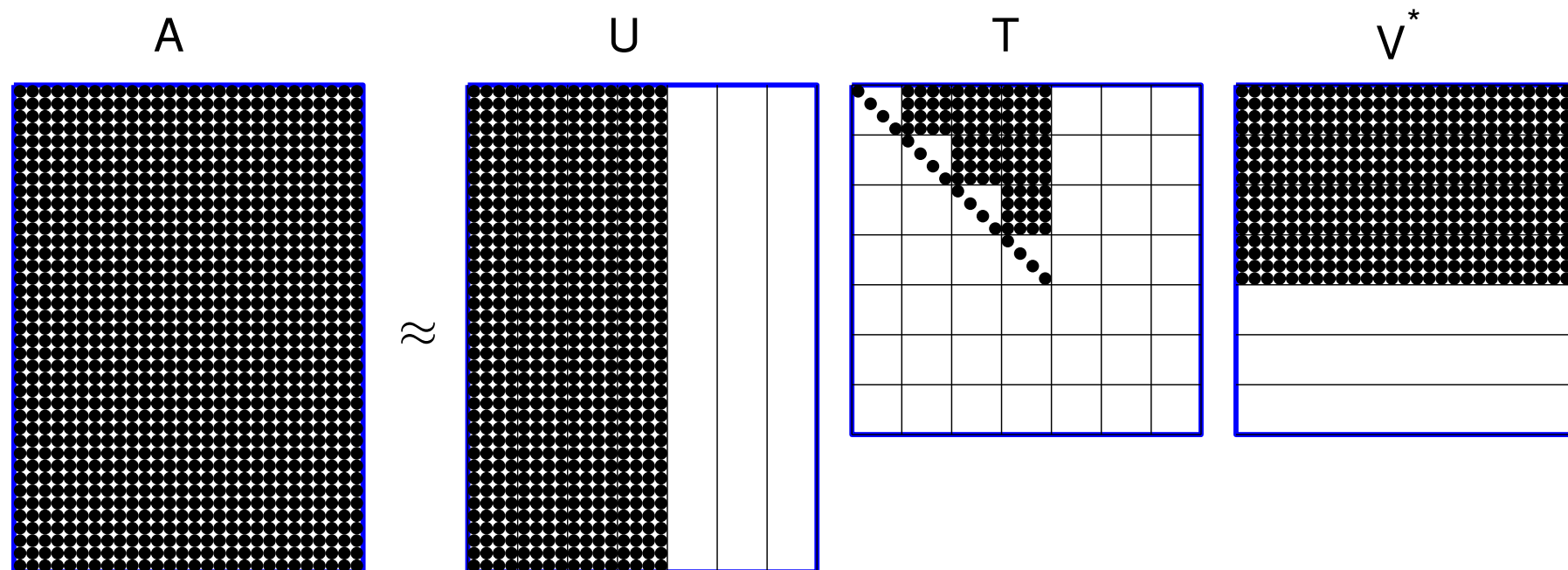
(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (efficient in practice): Build the factorization iteratively.

Illustration for block size $b = 4$:



The rank-16 approximation.

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

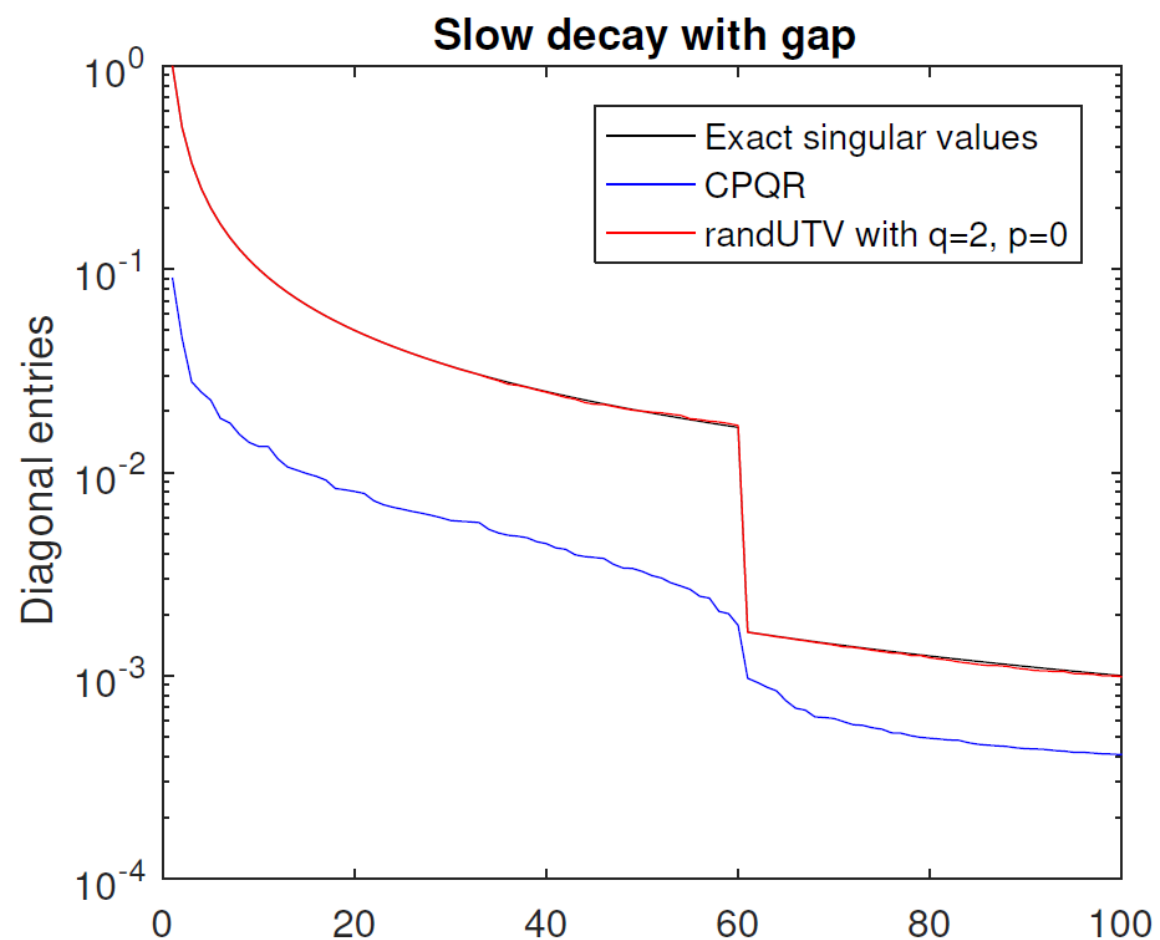
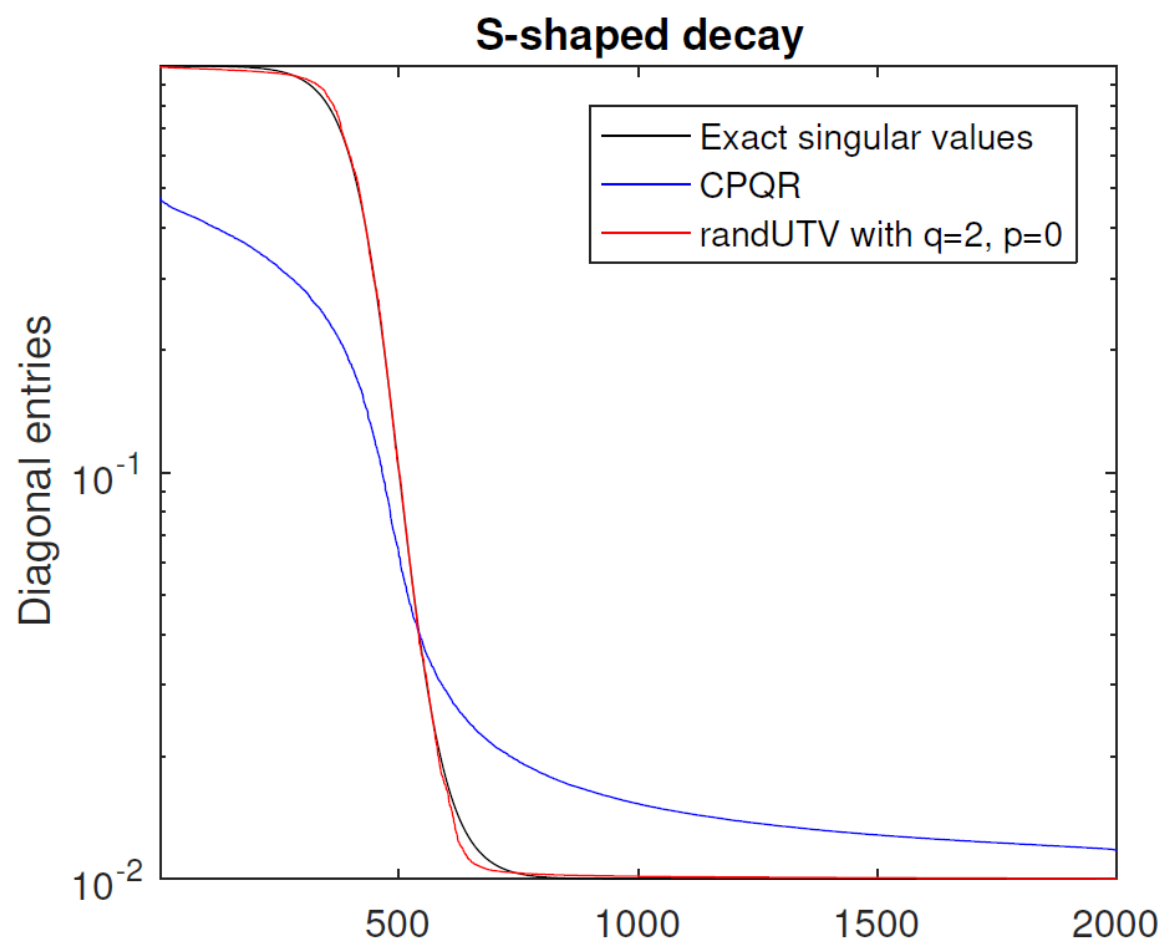
(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (efficient in practice): Build the factorization iteratively.

Numerical results: Exact singular values of \mathbf{A} vs. diagonal values of \mathbf{T} (in $\mathbf{A} = \mathbf{UTV}^$).*



Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Answer (best I know of): “**IterativeCUR**” Instead of building orthonormal basis, use “natural” bases formed by subsets of the rows and columns.

- The method is incremental, and “blocked” for computational efficiency.
- After each step of the iteration is completed, an a posteriori error estimation is used to provide a highly reliable stopping criterion.
- The “new” indices that are added at each step of the iteration are chosen using a sketch of the present residual $\mathbf{A} - \mathbf{CUR}$.
- DOWDATING of sketch is used to ensure that only $O(1)$ random samples from the matrix are required. (Say $b = 10$ or $b = 25$ samples.)
- After the initial sketch, *we never again visit the entire matrix*; instead, we only extract the chosen columns and rows.
- Overall complexity is $O(mn + (m + n)k^2)$ where k denotes the computed rank.

NEW! N. Pritchard, T. Park, Y. Nakatsukasa, PG. Martinsson, arxiv:2509.21963

Randomized SVD:

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** $\mathbf{\Omega}$.

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What if you do not know k in advance?

Note: Similar ideas can be used to build a *column-pivoted QR decomposition*.

The key benefit here is that you move essentially all flops into **BLAS3** operations!

This does not compromise the quality of the choice of pivots.

Resolution to long-standing problem in numerical linear algebra.

References: Martinsson *arXiv:1505.08115*; Duersch/Gu *arXiv:1509.06820*; Martinsson/Quintana-Ortí/Heavner/van de Geijn

SISC 2017; Duersch/Gu SISC 2017 and SIREV 2020.

Randomized pivoting in Householder QR



Speedup attained by our randomized algorithm HQRRP for computing a full column pivoted QR factorization of an $n \times n$ matrix. The speed-up is measured versus LAPACK's faster routine `dgeqp3` as implemented in Netlib (left) and Intel's MKL (right). Our implementation was done in C, and was executed on an Intel Xeon E5-2695. Joint work with G. Quintana-Ortí, N. Heavner, and R. van de Geijn. Available at: <https://github.com/flame/hqrrp/>

References: Martinsson *arXiv:1505.08115*; Duersch/Gu *arXiv:1509.06820*; Martinsson/Quintana-Ortí/Heavner/van de Geijn

SISC 2017; Duersch/Gu *SISC 2017 and SIREV 2020*.

Randomized SVD: Double-sided approximation

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** Ω .

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\Omega$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Recall that the output of the RSVD is the matrix

$$\mathbf{A}_{\text{approx}} = \mathbf{UDV}^* = \mathbf{Q}\mathbf{Q}^*\mathbf{A},$$

where \mathbf{Q} holds the basis for $\text{ran}(\mathbf{A})$ from the randomized range finder in steps (1)–(3).

Since $\mathbf{Q}\mathbf{Q}^*$ is the orthogonal projector onto $\text{ran}(\mathbf{A}\Omega)$, we can rewrite $\mathbf{A}_{\text{approx}}$ as:

$$\mathbf{A}_{\text{approx}} = \mathbf{Q}\mathbf{Q}^*\mathbf{A} = (\mathbf{A}\Omega)(\mathbf{A}\Omega)^\dagger\mathbf{A}.$$

Question: Can we sketch \mathbf{A} from *both sides*?

Randomized SVD: Double-sided approximation

Input: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p (say $p = 5$).

Output: Rank- $(k + p)$ factors \mathbf{U} , \mathbf{D} , and \mathbf{V} in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.

(1) Draw an $n \times (k + p)$ **random matrix** Ω .

(2) Form the $m \times (k + p)$ **sample matrix** $\mathbf{Y} = \mathbf{A}\Omega$.

(3) Compute an **ON matrix** \mathbf{Q} s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Recall that the output of the RSVD is the matrix

$$\mathbf{A}_{\text{approx}} = \mathbf{UDV}^* = \mathbf{Q}\mathbf{Q}^* \mathbf{A},$$

where \mathbf{Q} holds the basis for $\text{ran}(\mathbf{A})$ from the randomized range finder in steps (1)–(3).

Since $\mathbf{Q}\mathbf{Q}^*$ is the orthogonal projector onto $\text{ran}(\mathbf{A}\Omega)$, we can rewrite $\mathbf{A}_{\text{approx}}$ as:

$$\mathbf{A}_{\text{approx}} = \mathbf{Q}\mathbf{Q}^* \mathbf{A} = (\mathbf{A}\Omega) (\mathbf{A}\Omega)^\dagger \mathbf{A}.$$

Question: Can we sketch \mathbf{A} from *both sides*?

Answer: Yes; simply draw a sketch $\Psi\mathbf{A}$ of the row space as well. Then

$$\mathbf{A}_{\text{newapprox}} = (\mathbf{A}\Omega) (\mathbf{A}\Omega)^\dagger \mathbf{A} (\Psi\mathbf{A})^\dagger (\Psi\mathbf{A}) = \dots = (\mathbf{A}\Omega) (\Psi\mathbf{A}\Omega)^\dagger (\Psi\mathbf{A}).$$

(Formula goes back at least to Wedderburn 1934.)

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

Observation 1:

The matrix $\mathbf{A}_{\text{approx}}$ can be built *in a single pass over \mathbf{A}* .

In other words, we need to view each matrix entry of \mathbf{A} only once.

This cannot be done using deterministic methods (as far as I know).

“Streaming” or “single-view” algorithm.

Note: Using different over-sampling parameters p and p' for the column and row spaces is often better.

References: Alon, Gibbons, Matias and Szegedy (2002); Woolfe, Liberty, Rokhlin, and Tygert (2008); Clarkson and Woodruff (2009); Li, Nguyen and Woodruff (2014); Boutsidi, Woodruff and Zhong (2016), Tropp, Yurtsever, Udell and Cevher (2017); Pourkamali-Anaraki and Becker (2019); Wang, Gittens and Mahoney (2019); Nakatsukasa, *arXiv:2009.11392*, 2020; Dong & Martinsson, *arXiv:2104.05877*, 2021; ... many more ...

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ requires $O(mnk)$ flops.

$O(mnk)$ matches the flop count of Gram-Schmidt, or a Krylov method applied to a dense matrix.

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that *$\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ can be evaluated using asymptotically fewer flops.*

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

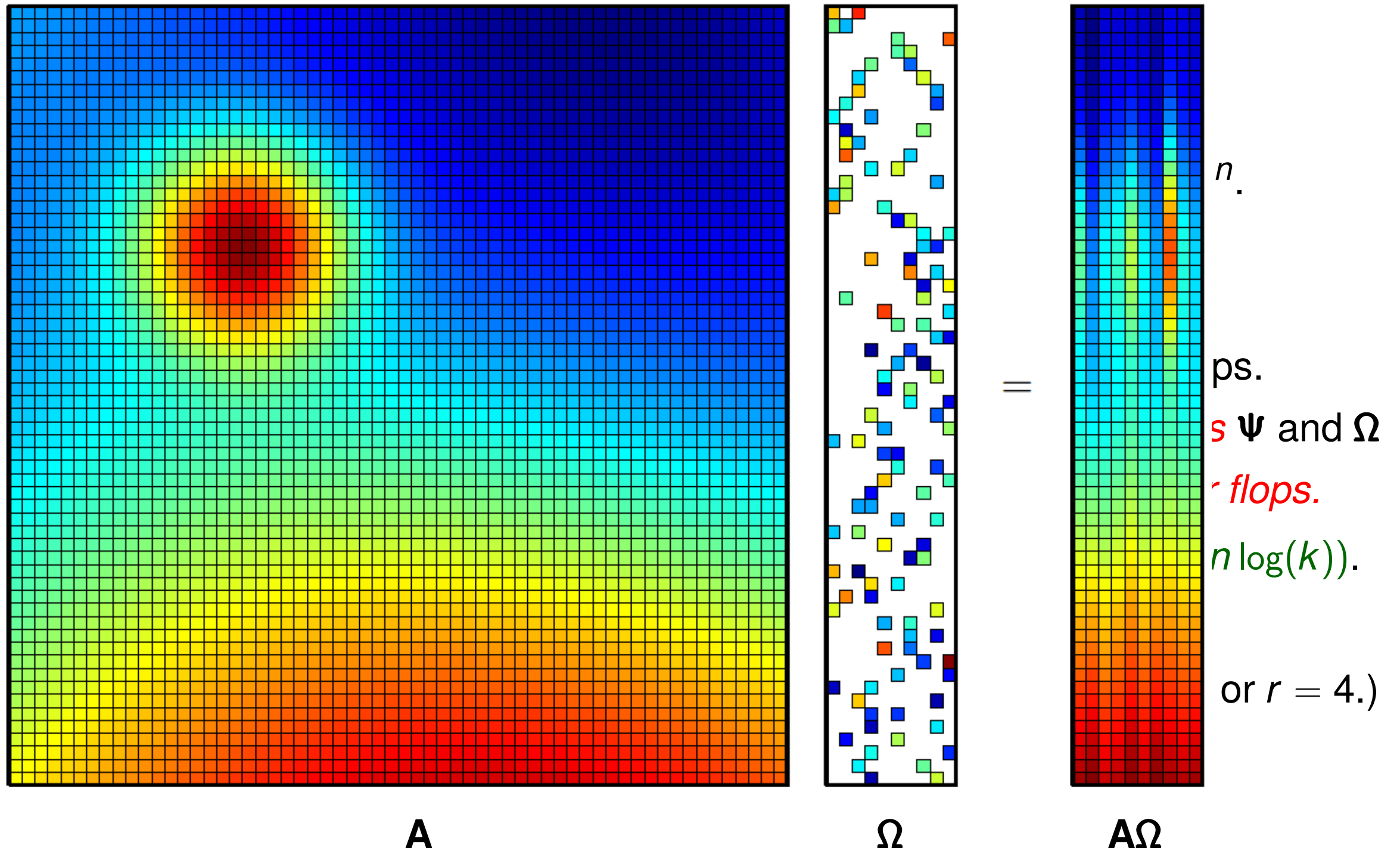
Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that *$\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ can be evaluated using asymptotically fewer flops.*

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn \log(k))$.
- Chains of Givens rotations (“Kac’s random walk”). Cost is $O(mn \log(k))$.
- “Sparse sign matrix”. Place r random entries in each row of $\mathbf{\Omega}$. (Say $r = 2$ or $r = 4$.)
Cost is now $O(mn)$!

Random
 Task: F
 Algorithm
 Form ap
 Observ
 Using G
 Instead
 with the
 • Ran
 • Cha
 • “Sp
 Cos



The matrix $\mathbf{\Omega}$ is a sparse random matrix. Two nonzero entries are placed randomly in each row. In consequence, each column of \mathbf{A} contributes to precisely two columns of the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$. This structured random map has $O(mn)$ complexity, is easy to work with practically, and often provides good accuracy.

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that *$\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ can be evaluated using asymptotically fewer flops.*

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn \log(k))$.
- Chains of Givens rotations (“Kac’s random walk”). Cost is $O(mn \log(k))$.
- “Sparse sign matrix”. Place r random entries in each row of $\mathbf{\Omega}$. (Say $r = 2$ or $r = 4$.)
Cost is now $O(mn)$!

When structured random matrices are used, cost can be reduced to $O(mn + (m + n)k^2)$.

Despite the pseudo-inverse, this can be done in a numerically stable way.

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that *$\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ can be evaluated using asymptotically fewer flops.*

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn \log(k))$.
- Chains of Given's rotations ("Kac's random walk"). Cost is $O(mn \log(k))$.
- "Sparse sign matrix". Place r random entries in each row of $\mathbf{\Omega}$. (Say $r = 2$ or $r = 4$.)

Cost is now $O(mn)$!

Theory is very weak, and typically requires $\ell = k + p \sim k \log(k)$.

In practice, $\ell \approx 2k$ is almost always more than enough.

A posteriori error estimation (a "*certificate of accuracy*") is very useful:

The idea is to draw a thin slice of a Gaussian, $\mathbf{G} \in \mathbb{R}^{n \times s}$ for $s = 10$ say, that is

quarantined from the construction of $\mathbf{A}_{\text{approx}}$ and is used purely to estimate the error

$$\|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}^2 \approx \frac{1}{s} \|(\mathbf{A} - \mathbf{A}_{\text{approx}})\mathbf{G}\|_{\text{F}}^2 = \frac{1}{s} \|(\mathbf{I} - \mathbf{P})\mathbf{A}\mathbf{G}\|_{\text{F}}^2.$$

Randomized SVD: Double-sided approximation

Task: Find a rank- k approximation to a given $m \times n$ matrix \mathbf{A} .

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p') \times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A}\mathbf{\Omega}) (\mathbf{\Psi}\mathbf{A}\mathbf{\Omega})^\dagger (\mathbf{\Psi}\mathbf{A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that *$\mathbf{A}\mathbf{\Omega}$ and $\mathbf{\Psi}\mathbf{A}$ can be evaluated using asymptotically fewer flops.*

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn \log(k))$.
- Chains of Given's rotations ("Kac's random walk"). Cost is $O(mn \log(k))$.
- "Sparse sign matrix". Place r random entries in each row of $\mathbf{\Omega}$. (Say $r = 2$ or $r = 4$.)

Cost is now $O(mn)$!

References: Ailon & Chazelle (2006); Liberty, Rokhlin, Tygert, and Woolfe (2006); Halko, Martinsson, Tropp (2011); Clarkson & Woodruff (2013); Meng & Mahoney (2013); Nelson & Nguyen (2013); Urano (2013); Nakatsukasa, *arXiv:2009.11392*, 2020; Dong & Martinsson, *arXiv:2104.05877*, 2021. Much subsequent work — "Fast Johnson-Lindenstrauss transform."

Random mixing for solving linear systems & least squares problems

Having access to *unitary* random maps opens the door other linear algebraic problems:

Computing full rank-revealing factorizations: Let \mathbf{A} be an $n \times n$ matrix. To build a *full* rank-revealing factorization of \mathbf{A} , draw Ω from a Haar distribution, and form

$$\mathbf{Y} = \mathbf{A}\Omega.$$

Then perform a QR factorization of \mathbf{Y} , so that

$$\mathbf{Y} = \mathbf{QR}.$$

One can prove that then a “rank-revealing” factorization is obtained through

$$\mathbf{A} = \mathbf{QR}\Omega^*.$$

The point here is **communication efficiency**. *(Demmel, Dumitriu, Holtz 2007)*

Setting $\mathbf{Y} = \mathbf{AA}^*\mathbf{A}\Omega$ is even better! *(Heavner, Chen, Gopal, Martinsson 2023)*

Solving linear systems without pivoting: Consider

$$\mathbf{Ax} = \mathbf{b}.$$

Randomized preconditioning results in a system that can be solved *without pivoting*

$$(\Psi^*\mathbf{A}\Omega) (\Omega^*\mathbf{x}) = \Psi^*\mathbf{b}.$$

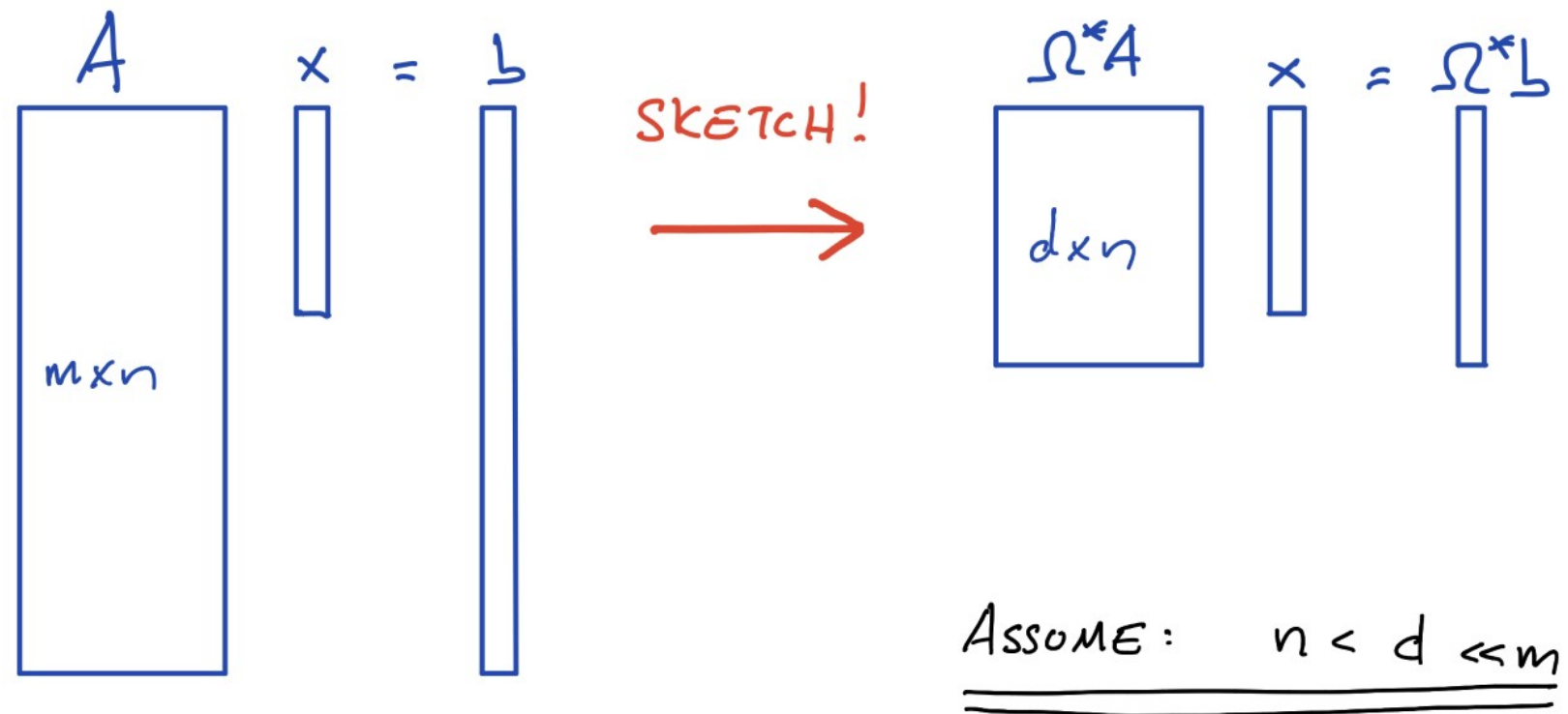
Again, the point is **communication efficiency**.

(Parker 1995)

Random mixing for solving linear systems & least squares problems

Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|$.

Draw a random embedding $\Omega \in \mathbb{R}^{m \times d}$ and construct a smaller *sketched* system.



A bold approach — “sketch-to-solve”:

Find the vector \mathbf{x} that solves the sketched system.

A safe approach — “sketch-to-precondition”:

Build a *preconditioner* $\mathbf{M} \in \mathbb{R}^{n \times n}$ by factorizing $\Omega^* \mathbf{A}$ so that $\Omega^* \mathbf{A} = \mathbf{QM}$.

Iterate on the preconditioned linear system $(\mathbf{AM}^{-1}) (\mathbf{Mx}) = \mathbf{b}$.

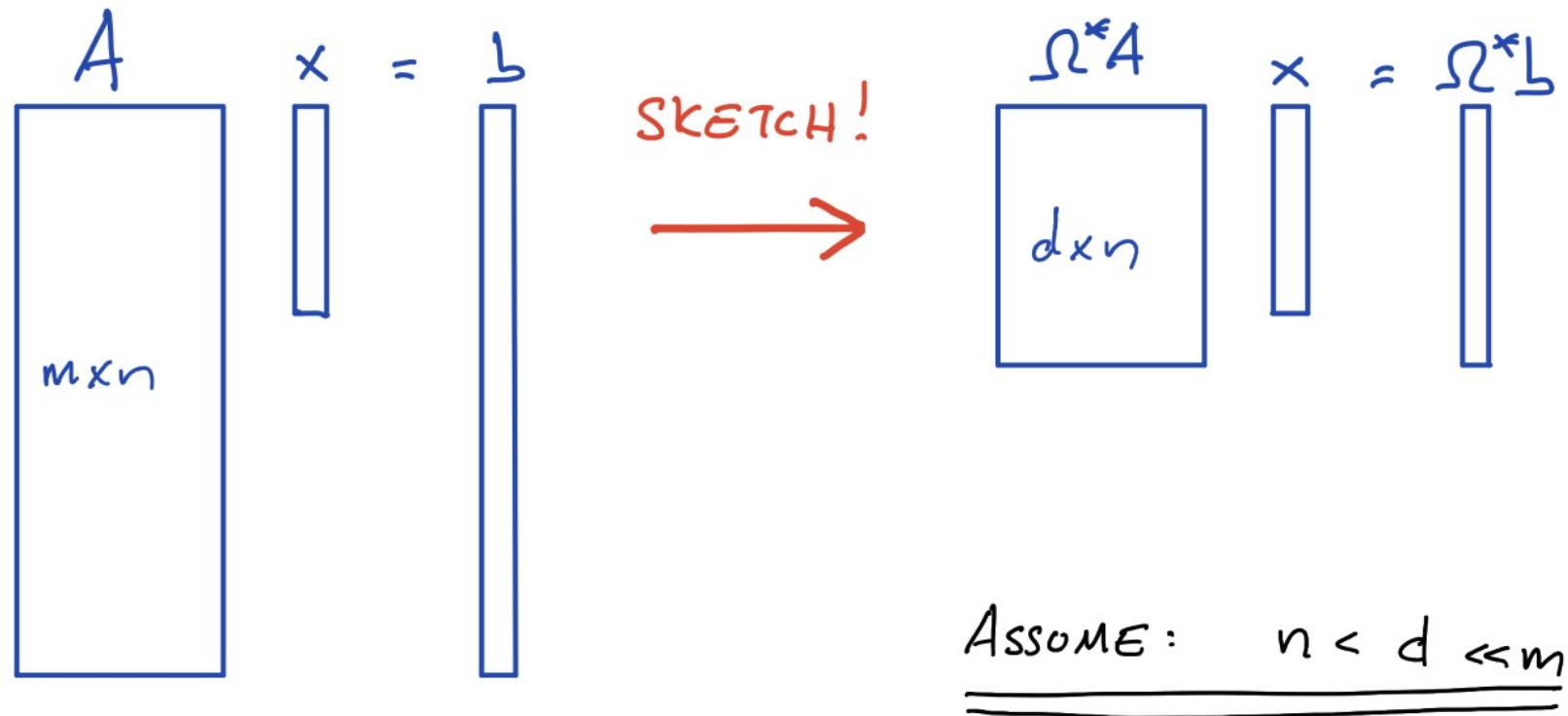
Ideal use of randomization. Guaranteed accuracy if you evaluate the residual.

Rokhlin/Tygert (2008), Avron/Maymounkov/Toledo (2010), many more

Random mixing for solving linear systems & least squares problems

Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|$.

Draw a random embedding $\Omega \in \mathbb{R}^{m \times d}$ and construct a smaller *sketched* system.

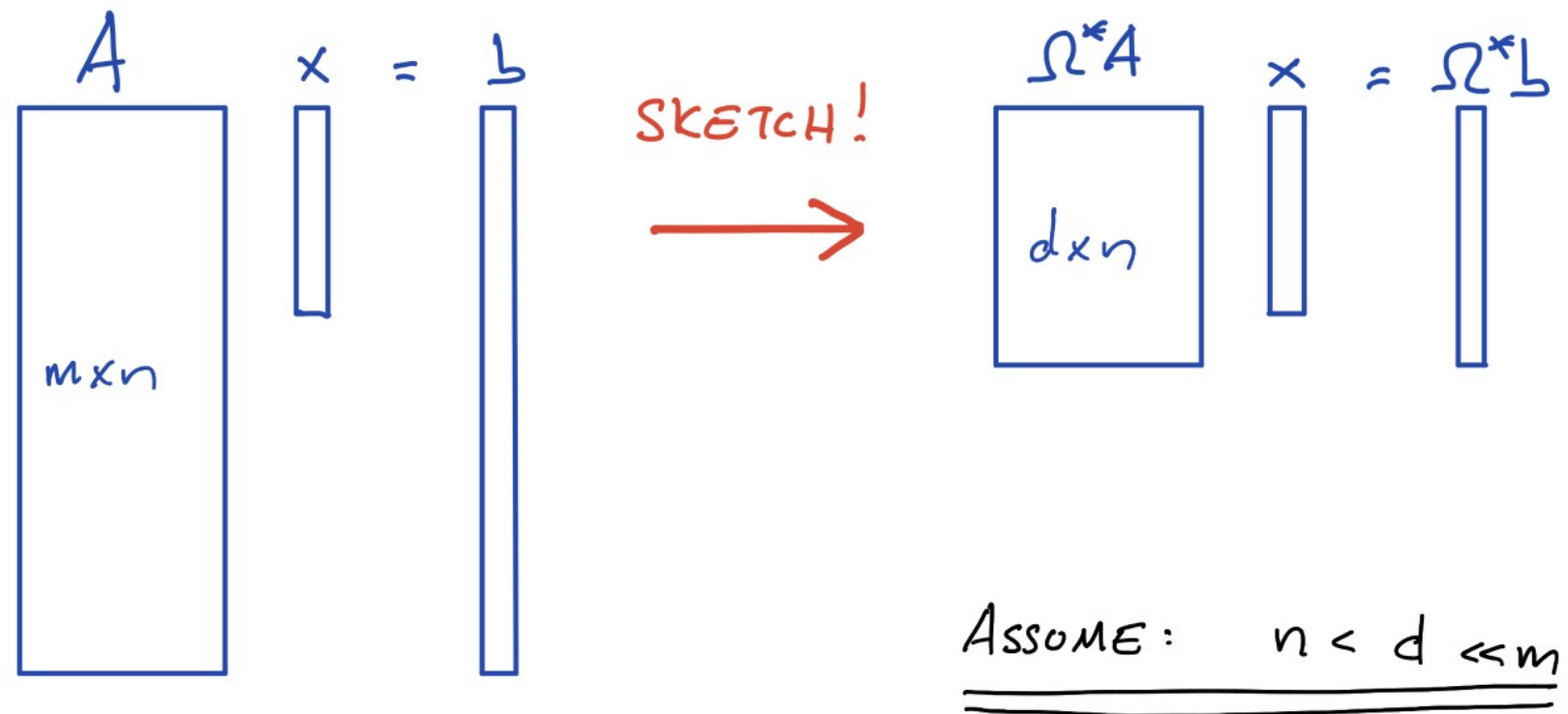


Question: Can the sketch be *down-sampling*? Simply pick d equations randomly?

Random mixing for solving linear systems & least squares problems

Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|$.

Draw a random embedding $\Omega \in \mathbb{R}^{m \times d}$ and construct a smaller *sketched* system.



Question: Can the sketch be *down-sampling*? Simply pick d equations randomly?

No, in general. Can fail catastrophically.

Yes, if you first randomly mix the equations.

Note: There are situations where randomized sampling is an essential tool – primarily when the whole matrix cannot be assembled. Examples include kernel ridge regression and certain gigantic linear systems arising in electronic structure calculations. “Down-sample and solve” is unavoidable in such cases.

It can also be very helpful when \mathbf{A} has structure, as in tensor approximation.

OUTLINE

- **High accuracy methods for linear algebraic problems**

- Low rank approximation: The randomized SVD. *Key success story*.
- Streaming and single-pass methods.
- Structured random maps (“fast Johnson-Lindenstrauss transforms”).
- Linear solvers: Sketch-to-solve vs. Sketch-to-precondition.
- Las Vegas style methods: Extremely robust and reliable.

- **Sampling based methods for huge problems**

- Monte Carlo style methods → less reliable, less accurate, less robust.
- Enable the solution of stupendously large problems that would otherwise be intractable.

- **Randomized methods for approximating continuum operators**

- Approximation of global operators of mathematical physics (solution operators, DtNs, ...).
- Tools for matrices that are not of global low rank, but have structure that can be exploited.
- Conceptually related to Fast Multipole Methods, Fast Direct Solvers, Calderón-Zygmund theory ...

Matrix approximation by sampling

Suppose that $\mathbf{A} = \sum_{t=1}^T \mathbf{A}_t$ where each \mathbf{A}_t is “simple” in some sense.

Matrix approximation by sampling

Suppose that $\mathbf{A} = \sum_{t=1}^T \mathbf{A}_t$ where each \mathbf{A}_t is “simple” in some sense.

Example: Sparse matrix written as a sum over its nonzero entries

$$\underbrace{\begin{bmatrix} 5 & -2 & 0 \\ 0 & 0 & -3 \\ 1 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}} = \underbrace{\begin{bmatrix} 5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_1} + \underbrace{\begin{bmatrix} 0 & -2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_2} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_3} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_4}$$

Example: Each \mathbf{A}_i could be a column of the matrix

$$\underbrace{\begin{bmatrix} 5 & -2 & 7 \\ 1 & 3 & -3 \\ 1 & -1 & 1 \end{bmatrix}}_{=\mathbf{A}} = \underbrace{\begin{bmatrix} 5 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_1} + \underbrace{\begin{bmatrix} 0 & -2 & 0 \\ 0 & 3 & 0 \\ 0 & -1 & 0 \end{bmatrix}}_{=\mathbf{A}_2} + \underbrace{\begin{bmatrix} 0 & 0 & 7 \\ 0 & 0 & -3 \\ 0 & 0 & 1 \end{bmatrix}}_{=\mathbf{A}_3}.$$

Example: Matrix-matrix multiplication broken up as a sum of rank-1 matrices:

$$\mathbf{A} = \mathbf{BC} = \sum_t \mathbf{B}(:, t) \mathbf{C}(t, :).$$

Matrix approximation by sampling

Suppose that $\mathbf{A} = \sum_{t=1}^T \mathbf{A}_t$ where each \mathbf{A}_t is “simple” in some sense.

Let $\{p_t\}_{t=1}^T$ be a probability distribution on the index vector $\{1, 2, \dots, T\}$.

Draw an index $t \in \{1, 2, \dots, T\}$ according to the probability distribution given, and set

$$\mathbf{X} = \frac{1}{p_t} \mathbf{A}_t.$$

Then from the definition of the expectation, we have

$$\mathbb{E}[\mathbf{X}] = \sum_{t=1}^T p_t \times \frac{1}{p_t} \mathbf{A}_t = \sum_{t=1}^T \mathbf{A}_t = \mathbf{A},$$

so \mathbf{X} is an *unbiased estimate* of \mathbf{A} .

Clearly, a single draw is not a good approximation — unrepresentative, *large variance*.

Instead, draw several samples and average:

$$\bar{\mathbf{X}} = \frac{1}{k} \sum_{t=1}^k \mathbf{X}_t,$$

where \mathbf{X}_t are independent samples from the same distribution.

As k grows, the variance will decrease, as usual. Various Bernstein inequalities apply.

Matrix approximation by sampling

As an illustration of the theory, we cite a matrix-Bernstein result from J. Tropp (2015):

Theorem: Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. Construct a probability distribution for $\mathbf{X} \in \mathbb{R}^{m \times n}$ that satisfies

$$\mathbb{E}[\mathbf{X}] = \mathbf{A} \quad \text{and} \quad \|\mathbf{X}\| \leq R.$$

Define the per-sample second-moment: $v(\mathbf{X}) := \max\{\|\mathbb{E}[\mathbf{X}\mathbf{X}^*]\|, \|\mathbb{E}[\mathbf{X}^*\mathbf{X}]\|\}$.

Form the matrix sampling estimator: $\bar{\mathbf{X}}_k = \frac{1}{k} \sum_{t=1}^k \mathbf{X}_t$ where $\mathbf{X}_t \sim \mathbf{X}$ are iid.

$$\text{Then } \mathbb{E}\|\bar{\mathbf{X}}_k - \mathbf{A}\| \leq \sqrt{\frac{2v(\mathbf{X}) \log(m+n)}{k}} + \frac{2R \log(m+n)}{3k}.$$

$$\text{Furthermore, for all } s \geq 0: \mathbb{P}[\|\bar{\mathbf{X}}_k - \mathbf{A}\| \geq s] \leq (m+n) \exp\left(\frac{-ks^2/2}{v(\mathbf{X}) + 2Rs/3}\right).$$

Suppose that we want $\mathbb{E}\|\mathbf{A} - \bar{\mathbf{X}}\| \leq 2\epsilon$. The theorem says to pick

$$k \geq \max\left\{\frac{2v(\mathbf{X}) \log(m+n)}{\epsilon^2}, \frac{2R \log(m+n)}{3\epsilon}\right\}$$

In other words, the number k of samples should be proportional to both $v(\mathbf{X})$ and to the upper bound R .

The scaling $k \sim \frac{1}{\epsilon^2}$ is discouraging, and unavoidable (since error $\epsilon \sim 1/\sqrt{k}$).

Matrix approximation by sampling – key points:

Sampling based methods provide a path forwards for problems where traditional techniques are not viable. Examples of applications:

- Kernel matrices in data analysis. Matrices are dense, and you cannot afford to form the entire matrix. “Kernel ridge regression”.
- Kronecker product matrices that arise in tensor approximations.
- Vast linear systems arising in quantum chemistry.

In all these examples, there is reason to expect the data to be amenable to sampling.

Theory is well understood. Sampling based techniques provides a solution to certain problems that had proven intractable to traditional deterministic methods.

When they are viable, techniques based on randomized embeddings tend to be preferable; they yield higher accuracy, and less variability in the outcome. ***Except:***

Matrix approximation by sampling – key points:

Sampling based methods provide a path forwards for problems where traditional techniques are not viable. Examples of applications:

- Kernel matrices in data analysis. Matrices are dense, and you cannot afford to form the entire matrix. “Kernel ridge regression”.
- Kronecker product matrices that arise in tensor approximations.
- Vast linear systems arising in quantum chemistry.

In all these examples, there is reason to expect the data to be amenable to sampling.

Theory is well understood. Sampling based techniques provides a solution to certain problems that had proven intractable to traditional deterministic methods.

When they are viable, techniques based on randomized embeddings tend to be preferable; they yield higher accuracy, and less variability in the outcome. **Except:**

Sampling based methods excel at computing a *Cholesky factorization* of a symmetric positive definite matrix, since you can use the diagonal elements to form sampling probabilities. When combined with rejection sampling, sublinear complexity and highly accurate methods become possible. Topic of 2025 Boeing colloquium by Joel Tropp: [Epperly, Tropp, Webber, SIMAX, 2025] Extension to general matrices (through the Gram matrix) is also highly effective: [Dong, Chen, Martinsson, and Pearce, SIMAX, 2025]

OUTLINE

- **High accuracy methods for linear algebraic problems**

- Low rank approximation: The randomized SVD. *Key success story*.
- Streaming and single-pass methods.
- Structured random maps (“fast Johnson-Lindenstrauss transforms”).
- Linear solvers: Sketch-to-solve vs. Sketch-to-precondition.
- Las Vegas style methods: Extremely robust and reliable.

- **Sampling based methods for huge problems**

- Monte Carlo style methods → less reliable, less accurate, less robust.
- Enable the solution of stupendously large problems that would otherwise be intractable.

- **Randomized methods for approximating continuum operators**

- Approximation of global operators of mathematical physics (solution operators, DtNs, ...).
- Tools for matrices that are not of global low rank, but have structure that can be exploited.
- Conceptually related to Fast Multipole Methods, Fast Direct Solvers, Calderón-Zygmund theory ...

Global operators in scientific computing

A key challenge in scientific computing is that many of the operators we seek to model computationally are *global* in the sense that every point in the domain talks to every other point. In consequence, they need to be represented as *dense* matrices.

Examples:

- Solution operators of elliptic PDEs.
- Boundary-to-boundary operators (e.g. Dirichlet-to-Neumann).
- Time-evolution operators of parabolic PDEs.
- Scattering matrices for many wave propagation problems.
- Schur complements in sparse direct solvers.

Algorithms:

- Methods for *applying* global operators rapidly are well established in specialized cases (FFT, Fast Multipole Methods etc). Methods exist for more general operators, but this remains a topic of research (e.g. \mathcal{H} -matrices, randomized compression).
- Techniques for *inverting, factorizing, exponentiating, ...* such operators exist, with progress ongoing. “Fast Direct Solvers”.
- Our focus today: Methods for obtaining the data sparse representation.

Global operators in scientific computing

A key challenge in scientific computing is that many of the operators we seek to model computationally are *global* in the sense that every point in the domain talks to every other point. In consequence, they need to be represented as *dense* matrices.

Examples:

- Solution operators of elliptic PDEs.
- Boundary-to-boundary operators (e.g. Dirichlet-to-Neumann).
- Time-evolution operators of parabolic PDEs.
- Scattering matrices for many wave propagation problems.
- Schur complements in sparse direct solvers.

Vision:

Create a framework where we can explicitly form data sparse representations of global linear operators, use them to model physical systems, and then *directly* solving the resulting equilibrium equations.

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma, \end{cases}$$

where Ω is a domain (2D or 3D) with boundary Γ , and where A is a linear elliptic differential operator; possibly with variable coefficients.

Examples of problems:

- The equations of linear elasticity.
- Stokes' equation.
- Helmholtz' equation (at least at low and intermediate frequencies).
- Time-harmonic Maxwell (at least at low and intermediate frequencies).

Archetypical example: Poisson equation with Dirichlet boundary data:

$$\begin{cases} -\Delta u(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma. \end{cases}$$

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma, \end{cases}$$

where Ω is a domain (2D or 3D) with boundary Γ , and where A is a linear elliptic differential operator; possibly with variable coefficients.

Examples of problems:

- The equations of linear elasticity.
- Stokes' equation.
- Helmholtz' equation (at least at low and intermediate frequencies).
- Time-harmonic Maxwell (at least at low and intermediate frequencies).

Archetypical example: Poisson equation with Dirichlet boundary data:

$$\begin{cases} -\Delta u(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma. \end{cases}$$

Standard numerical recipe for (BVP): (1) Discretize via FD/FEM. (2) Iterative solver.

Our point of interest: The solution operator for (BVP).

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma, \end{cases}$$

where Ω is a domain (2D or 3D) with boundary Γ , and where A is a linear elliptic differential operator; possibly with variable coefficients.

Linear solution operators: As a warmup, let us consider the Poisson equation

$$-\Delta u(\mathbf{x}) = g(\mathbf{x}) \quad \mathbf{x} \in \mathbb{R}^2$$

(with suitable decay conditions at infinity to ensure uniqueness). The solution is given by

$$(SLN) \quad u(\mathbf{x}) = \int_{\mathbb{R}^2} \phi(\mathbf{x} - \mathbf{y}) g(\mathbf{y}) d\mathbf{y}, \quad \mathbf{x} \in \mathbb{R}^2.$$

where the “fundamental solution” of the Laplace operator $-\Delta$ on \mathbb{R}^2 is defined by

$$\phi(\mathbf{x}) = -\frac{1}{2\pi} \log |\mathbf{x}|.$$

In principle very simple. Numerically non-trivial, however: The operator is *global*, so discretizing it leads to a *dense* matrix. (There is also the singular kernel to worry about!)

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma, \end{cases}$$

where Ω is a domain (2D or 3D) with boundary Γ , and where A is a linear elliptic differential operator; possibly with variable coefficients.

Linear solution operators: A general solution operator for (BVP) takes the form

$$(SLN) \quad u(\mathbf{x}) = \int_{\Omega} G(\mathbf{x}, \mathbf{y}) g(\mathbf{y}) d\mathbf{y} + \int_{\Gamma} F(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) dS(\mathbf{y}), \quad \mathbf{x} \in \Omega,$$

where G and F are two kernel functions that depend on A , B , and Ω .

Good: The operators in (SLN) are friendly and nice.

Bounded, smoothing, often fairly stable, etc.

Bad: The kernels G and F in (SLN) are generally *unknown*.

(Other than in trivial cases — constant coefficients and very simple domains.)

Bad: The operators in (SLN) are *global*.

Dense matrices upon discretization. $O(N^2)$ cost? $O(N^3)$ cost?

Example: Solution operator to an elliptic PDE

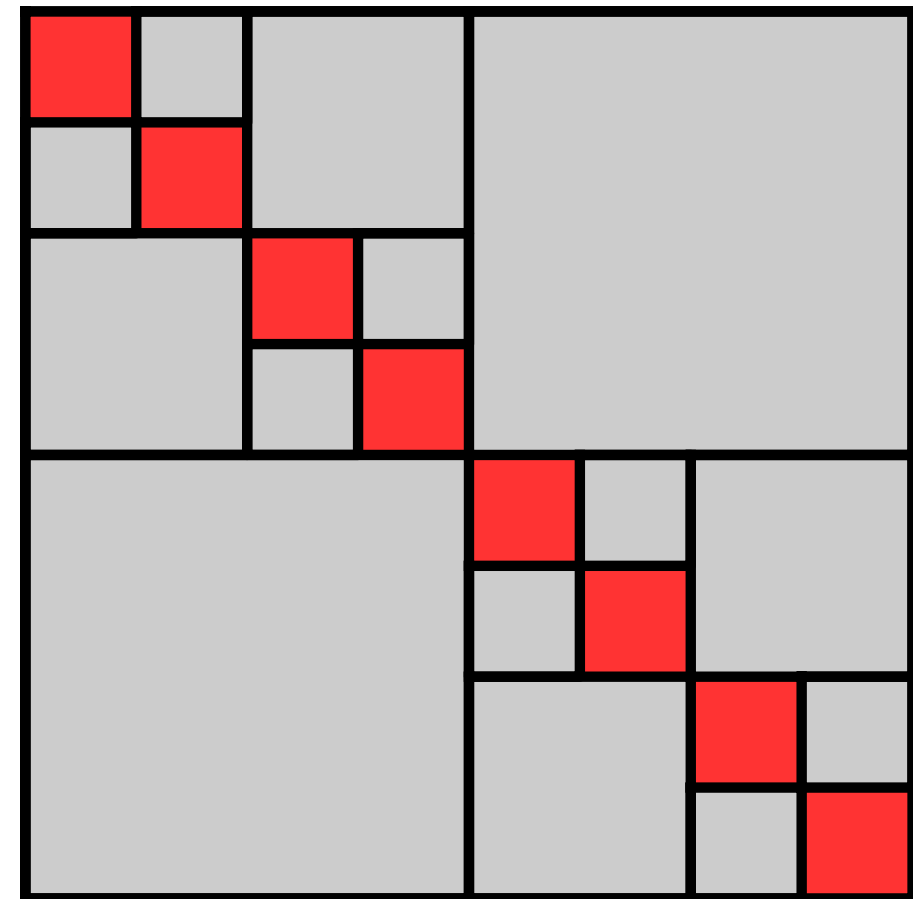
Consider a linear boundary value problem of the form

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma, \end{cases}$$

where Ω is a domain (2D or 3D) with boundary Γ , and where A is a linear elliptic differential operator; possibly with variable coefficients.

Recurring idea: Upon discretization, (SLN) leads to a matrix with *off-diagonal blocks of low numerical rank*.

This property can be exploited to attain linear or close to linear complexity for operations such as matrix-vector multiply, matrix-matrix multiply, LU factorization, matrix inversion, forming of Schur complements, etc.



All gray blocks have low rank.

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma, \end{cases}$$

where Ω is a domain (2D or 3D) with boundary Γ , and where A is a linear elliptic differential operator; possibly with variable coefficients.

Strong connections to Calderón-Zygmund theory for singular integral operators.

References: Fast Multipole Method (Greengard, Rokhlin); Panel Clustering (Hackbusch); \mathcal{H} - and \mathcal{H}^2 -matrices (Hackbusch et al); Hierarchically Block Separable matrices; Hierarchically Semi Separable matrices (Xia et al); HODLR matrices (Darve et al); BLR matrices (Buttari, Amestoy, Mary, ...); ...

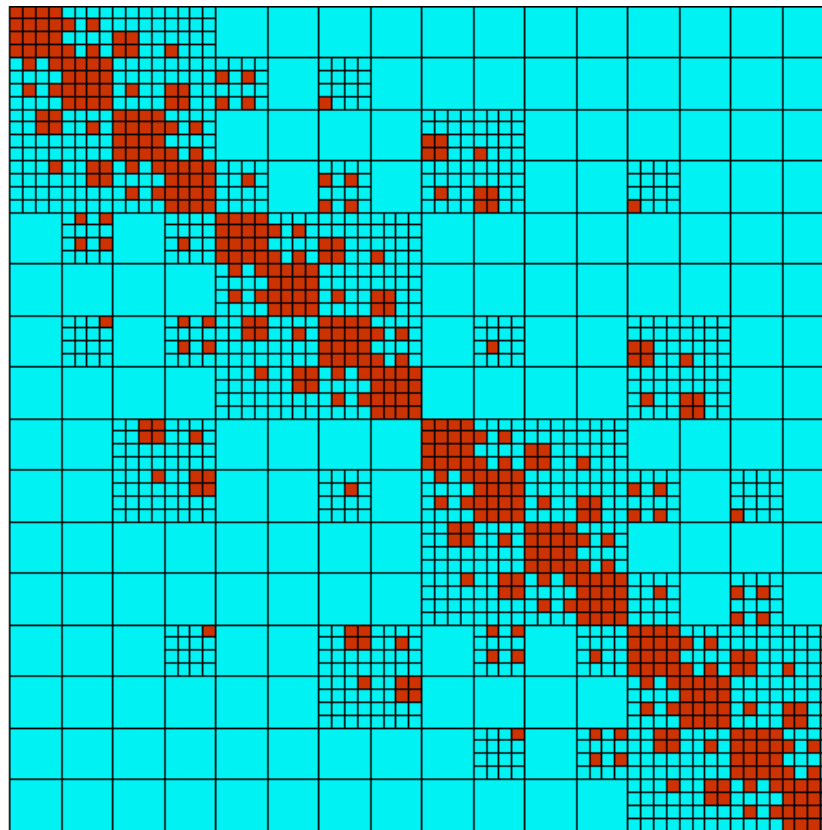
Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma, \end{cases}$$

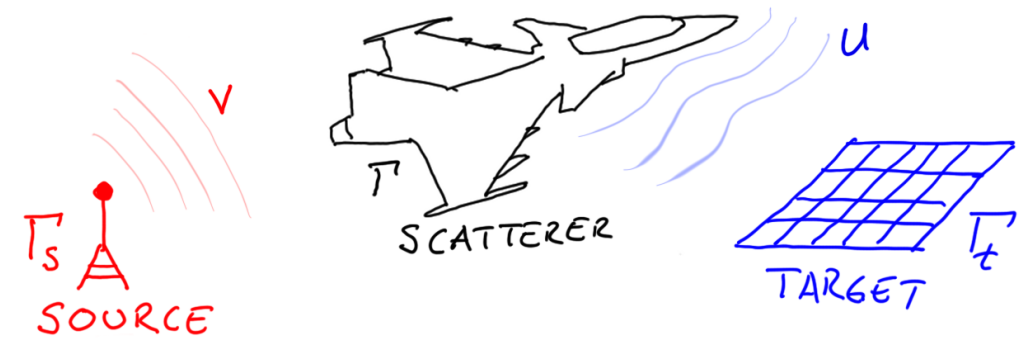
where Ω is a domain (2D or 3D) with boundary Γ , and where A is a linear elliptic differential operator; possibly with variable coefficients.

In real life, tessellation patterns of rank structured matrices tend to be more complex ...



Example: Boundary integral equation

Recall that many boundary value problems can advantageously be recast as *boundary integral equations*. Consider, e.g., (sound-soft) acoustic scattering from a finite body:



$$(2) \quad \begin{cases} -\Delta u(\mathbf{x}) - \kappa^2 u(\mathbf{x}) = 0 & \mathbf{x} \in \mathbb{R}^3 \setminus \bar{\Omega} \\ u(\mathbf{x}) = v(\mathbf{x}) & \mathbf{x} \in \partial\Omega \\ \lim_{|\mathbf{x}| \rightarrow \infty} |\mathbf{x}| (\partial_{|\mathbf{x}|} u(\mathbf{x}) - i\kappa u(\mathbf{x})) = 0. \end{cases}$$

The BVP (2) has an alternative mathematical formulation in the BIE

$$(3) \quad -\pi i \sigma(\mathbf{x}) + \int_{\partial\Omega} \left(\left(\partial_{\mathbf{n}(\mathbf{y})} + i\kappa \right) \frac{e^{i\kappa|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|} \right) \sigma(\mathbf{y}) dS(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega.$$

The integral equation (3) has several advantages over the PDE (2), including:

- The domain of computation $\partial\Omega$ is finite.
- The domain of computation $\partial\Omega$ is 2D, while $\mathbb{R}^3 \setminus \bar{\Omega}$ is 3D.
- Equation (3) is inherently well-conditioned (as a “2nd kind Fredholm equation”).

The integral operator (3) is global, and a matrix resulting from discretizing it is dense.

But both this matrix and its inverse are rank structured $\rightarrow O(N)$ solvers possible.

Example: Poincaré-Steklov operators (Dirichlet-to-Neumann, etc)

Consider a well posed linear boundary value problem

$$(BVP) \quad \begin{cases} Au(\mathbf{x}) = 0, & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma. \end{cases}$$

For $\mathbf{x} \in \Gamma$, let $n(\mathbf{x})$ denote the normal derivative of the solution u at \mathbf{x} . Then the map

$$T : f \mapsto n$$

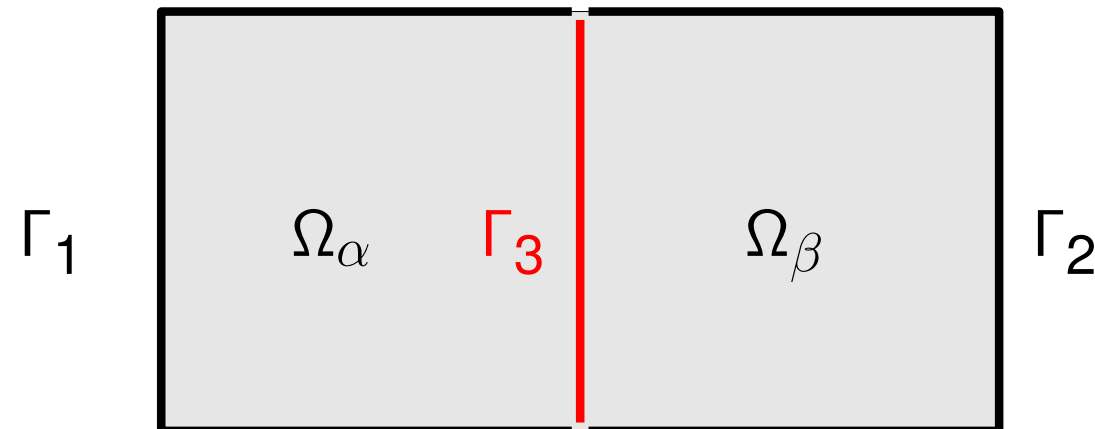
is known as the *Dirichlet-to-Neumann (DtN)* map.

The DtN is a powerful tool in many areas of scientific computing:

- It provides a compressed representation that “hides” interior dynamics in a subdomain from the rest of the model. A mathematically “ideal” reduced model.
- Essential tool for understanding domain decomposition methods.
- The basis of many methods for inverse problems where you seek to reconstruct variable coefficients in A by observing input-output pairs.
- Etc etc.

Example: Poincaré-Steklov operators (Dirichlet-to-Neumann, etc)

Let us consider a boundary value problem with Dirichlet data on a rectangular domain Ω partitioned into two subdomains $\Omega = \Omega_\alpha \cup \Omega_\beta$:



We partition the boundary of Ω as $\partial\Omega = \Gamma_1 \cup \Gamma_2$, and let Γ_3 denote the interior boundary. We know the Dirichlet data f_1 and f_2 on Γ_1 and Γ_2 , but we do *not* know it on the “artificial” boundary Γ_3 . However, if we know the DtN maps T^α and T^β for Ω_α and Ω_β , then we can decompose these as

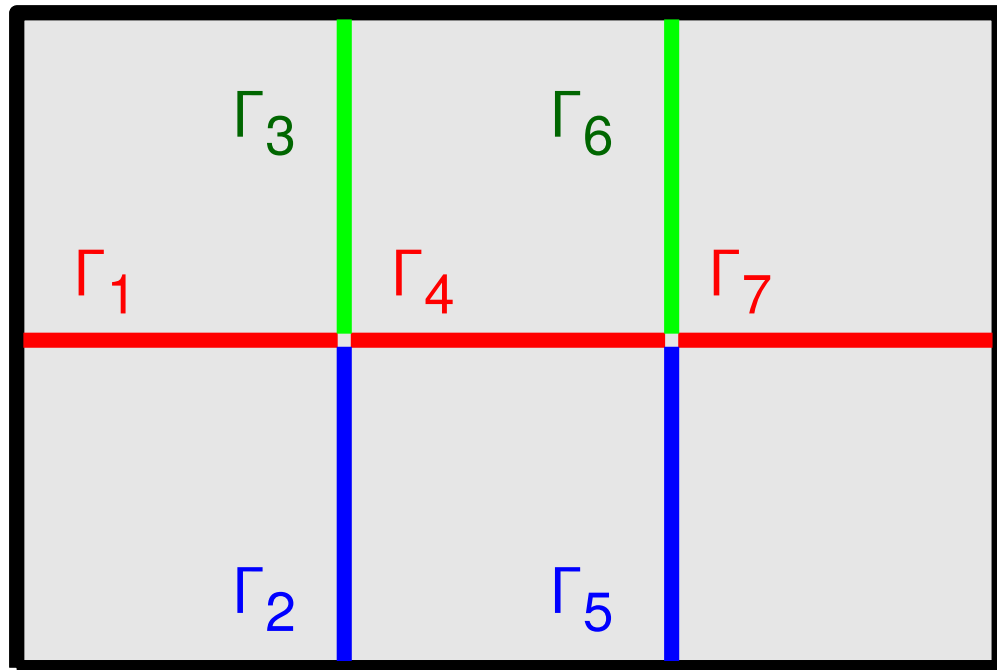
$$\begin{bmatrix} T_{11}^\alpha & T_{13}^\alpha \\ T_{31}^\alpha & T_{33}^\alpha \end{bmatrix} \begin{bmatrix} f_1 \\ f_3 \end{bmatrix} = \begin{bmatrix} n_1 \\ n_3^\alpha \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} T_{22}^\beta & T_{23}^\beta \\ T_{32}^\beta & T_{33}^\beta \end{bmatrix} \begin{bmatrix} f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} n_2 \\ n_3^\beta \end{bmatrix}$$

where n_3^α and n_3^β represent the boundary fluxes through Γ_3 from Ω_α and Ω_β , respectively. Since $n_3^\alpha + n_3^\beta = 0$, we can now form an equation for the unknown quantity f_3 as

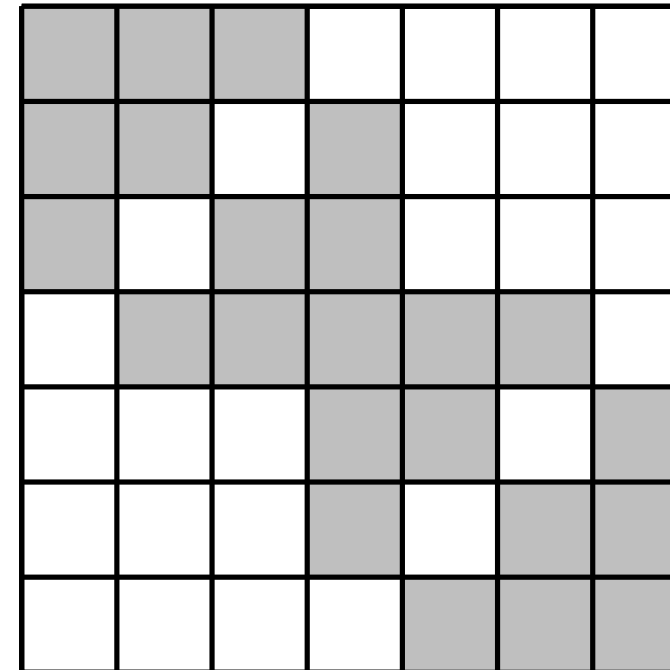
$$(T_{33}^\alpha + T_{33}^\beta) f_3 = -T_{31}^\alpha f_1 - T_{32}^\beta f_2.$$

Example: Poincaré-Steklov operators (Dirichlet-to-Neumann, etc)

Illustration of how the DtN map can be used to weld together six subdomains in a domain decomposition problem.



The domain.



The sparsity pattern of the linear system that determines the Dirichlet data on the interior boundaries $\{\Gamma_i\}_{i=1}^7$.

Example: Poincaré-Steklov operators (Dirichlet-to-Neumann, etc)

Consider the free space acoustic scattering problem

$$\begin{cases} -\Delta u(\mathbf{x}) - \kappa^2 (1 - b(\mathbf{x})) u(\mathbf{x}) = -\kappa^2 b(\mathbf{x}) v(\mathbf{x}), & \mathbf{x} \in \mathbb{R}^2 \\ \lim_{|\mathbf{x}| \rightarrow \infty} \sqrt{|\mathbf{x}|} (\partial_{|\mathbf{x}|} u(\mathbf{x}) - i\kappa u(\mathbf{x})) = 0, \end{cases}$$

where

- b is a smooth scattering potential with **compact support**, where
- v is a given “incoming potential” and where
- u is the sought “outgoing potential.”

Introduce an artificial box Ω such that $\text{support}(b) \subseteq \Omega$.

On Ω :

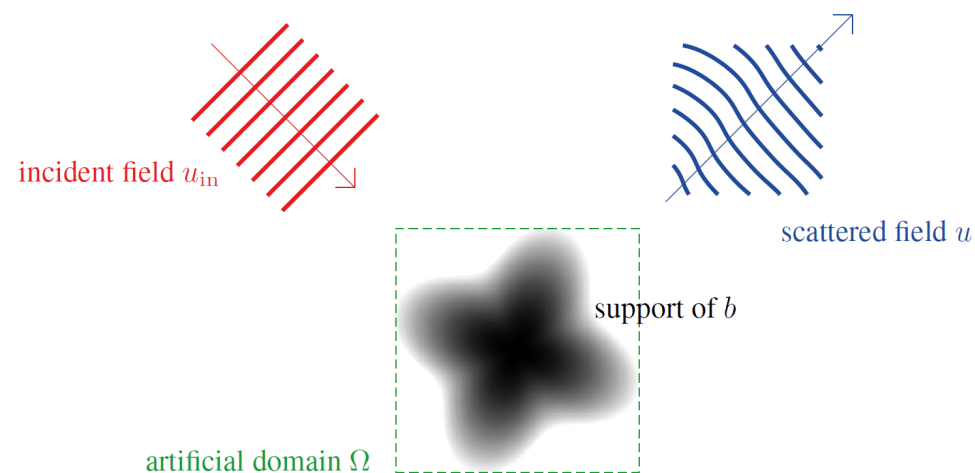
- Variable coefficient PDE.
- Discretize the PDE.
- Build DtN for $\partial\Omega$.

On Ω^c :

- Constant coefficient PDE.
- Use BIE.
- Build DtN for $\partial\Omega^c$.

Glue the domains together using the DtNs.

(Actually, *impedance-to-impedance (ItI)* maps are better.)



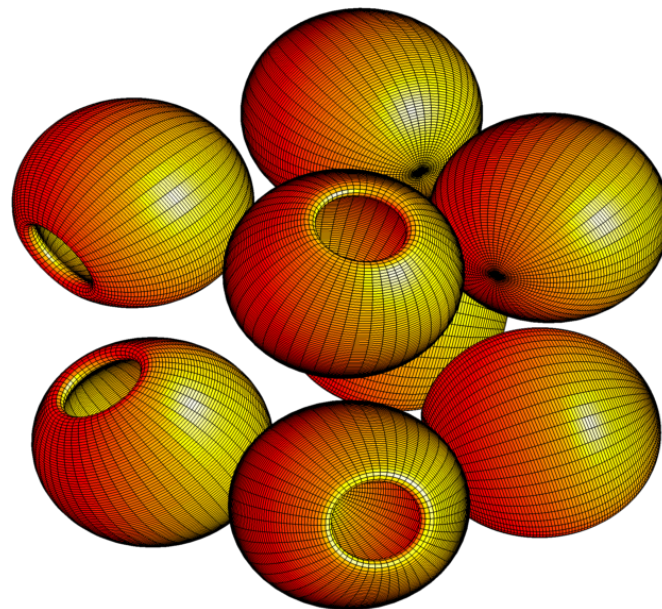
Example: Scattering operators

A *scattering operator* is the linear operator that maps an incoming wave to an scattered field in acoustic or electromagnetic scattering problems.

Scattering operators are powerful tools for solving multibody scattering problems, as they break a problem into smaller parts:

- A local computation is used to build a scattering operator for each individual body. These computations are unconnected, so highly parallelizable.
- Form a global system that uses the scattering matrices to describe how the bodies talk to each.

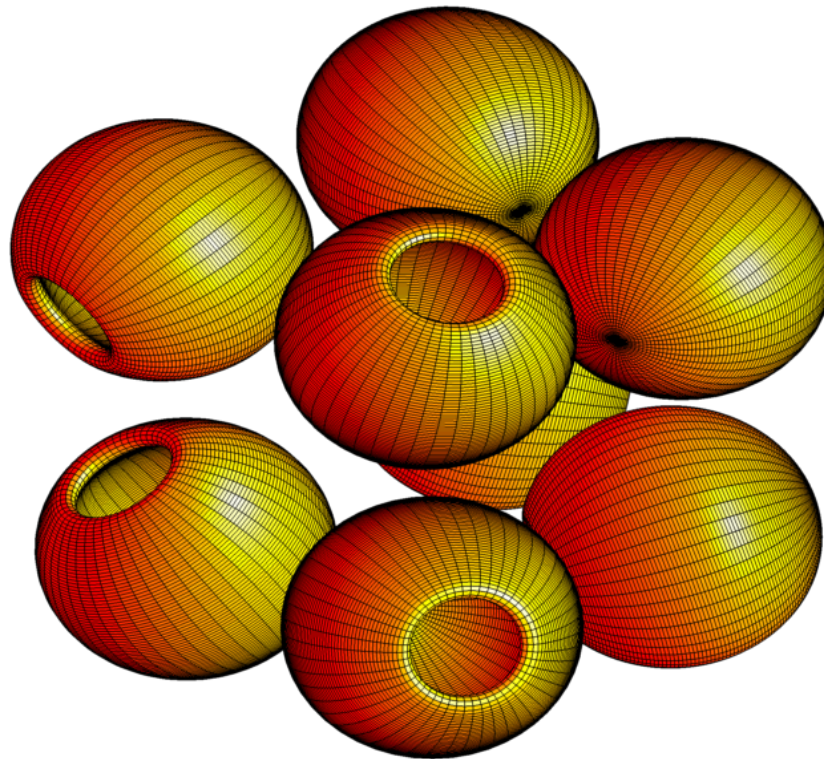
The benefit is that the global system you form this way is *far smaller* than a global system that fully resolves all individual scatters. (And often better conditioned too!)



Example: Scattering operators

A *scattering operator* is the linear operator that maps an incoming wave to an scattered field in acoustic or electromagnetic scattering problems.

Example: Acoustic scattering on the exterior domain. Each bowl is about 5λ .

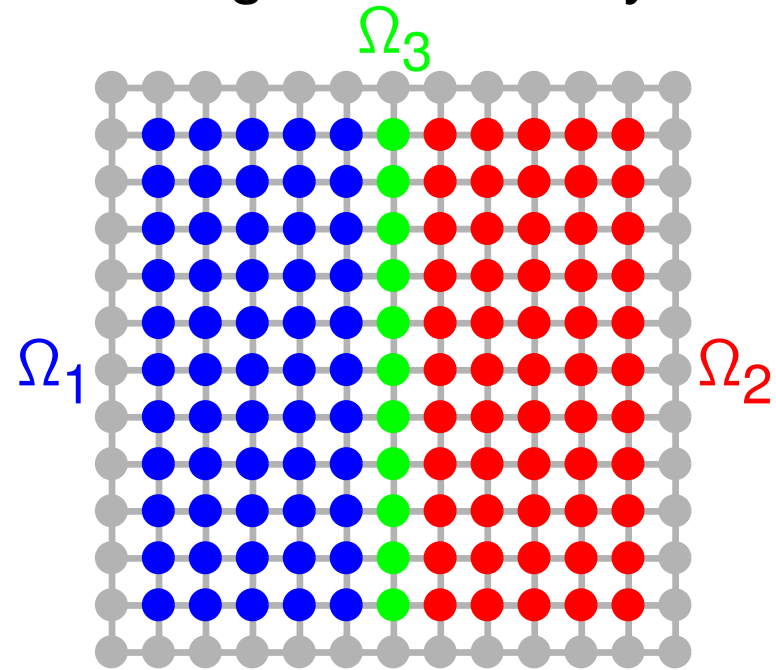


A hybrid direct/iterative solver is used (a highly accurate scattering matrix \mathbf{S}_i is computed for body i) to form a global system

$$(4) \quad \tilde{\mathbf{q}}_i + \mathbf{S}_{ii} \left(\sum_{j \neq i} \mathbf{A}_{ij} \tilde{\mathbf{q}}_j \right) = \mathbf{S}_{ii} \tilde{\mathbf{v}}_i, \quad i = 1, 2, \dots, J,$$

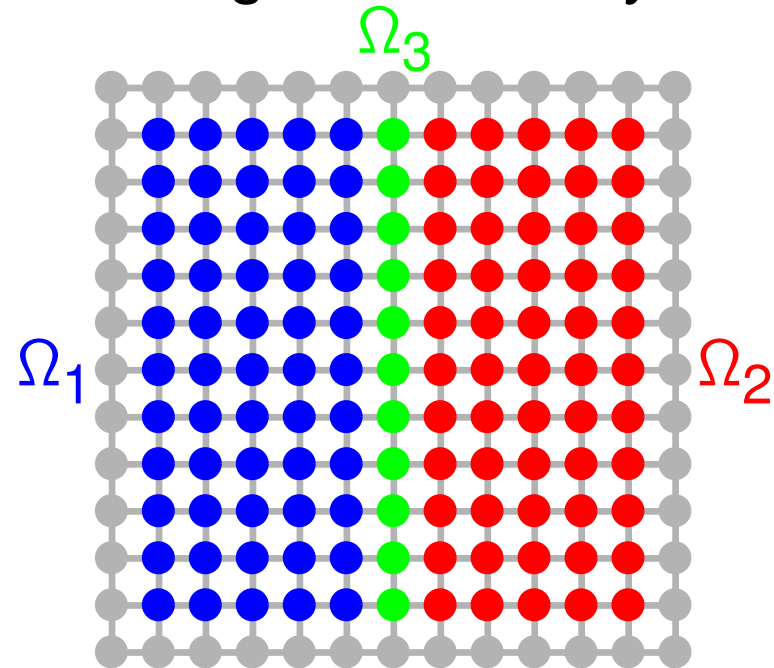
On an office desktop, we achieved an accuracy of 10^{-5} , in about 6h (essentially all the time is spent in applying the inter-body interactions via the Fast Multipole Method). Accuracy 10^{-7} took 27h.

Example: Schur complements Consider a finite difference discretization on a square resulting in a linear system $\mathbf{A}\mathbf{u} = \mathbf{b}$. Let us partition the nodes into three sets as follows:



$$\mathbf{A} = \begin{array}{|c|c|c|} \hline \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \\ \hline \end{array}$$

Example: Schur complements Consider a finite difference discretization on a square resulting in a linear system $\mathbf{A}\mathbf{u} = \mathbf{b}$. Let us partition the nodes into three sets as follows:



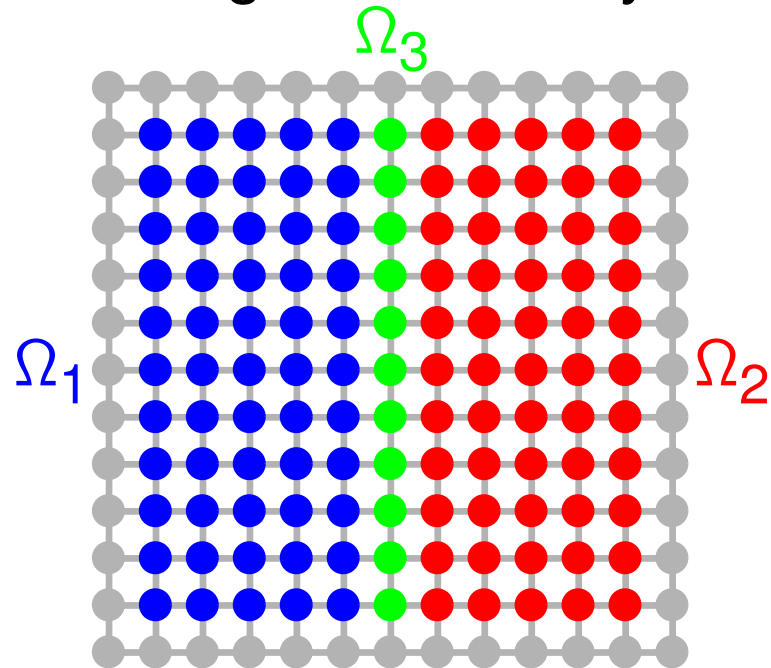
$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix}$$

Now suppose that we can somehow construct \mathbf{A}_{11}^{-1} and \mathbf{A}_{22}^{-1} . Then

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{A}_{31}\mathbf{A}_{11}^{-1} & \mathbf{A}_{32}\mathbf{A}_{22}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{A}_{11}^{-1}\mathbf{A}_{13} \\ \mathbf{0} & \mathbf{I} & \mathbf{A}_{22}^{-1}\mathbf{A}_{23} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement*.

Example: Schur complements Consider a finite difference discretization on a square resulting in a linear system $\mathbf{A}\mathbf{u} = \mathbf{b}$. Let us partition the nodes into three sets as follows:



$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix}$$

Now suppose that we can somehow construct \mathbf{A}_{11}^{-1} and \mathbf{A}_{22}^{-1} . Then

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{A}_{31}\mathbf{A}_{11}^{-1} & \mathbf{A}_{32}\mathbf{A}_{22}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{A}_{11}^{-1}\mathbf{A}_{13} \\ \mathbf{0} & \mathbf{I} & \mathbf{A}_{22}^{-1}\mathbf{A}_{23} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement*.

In other words, in order to invert \mathbf{A} , we need to execute three steps:

- Invert \mathbf{A}_{11} to form \mathbf{A}_{11}^{-1} .
- Invert \mathbf{A}_{22} to form \mathbf{A}_{22}^{-1} .
- Invert $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$.

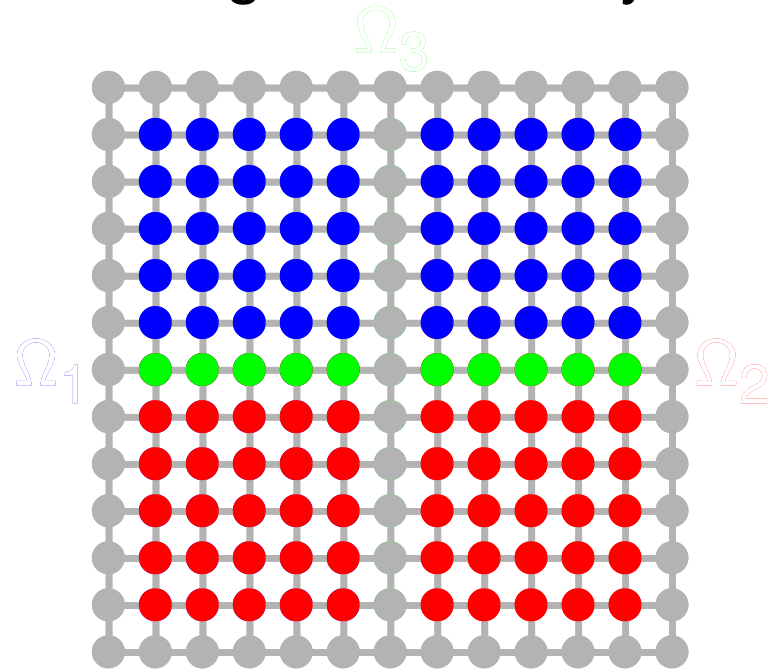
size $\sim N/2 \times N/2$

size $\sim N/2 \times N/2$

size $\sim \sqrt{N} \times \sqrt{N}$

Notice the obvious recursion!

Example: Schur complements Consider a finite difference discretization on a square resulting in a linear system $\mathbf{A}\mathbf{u} = \mathbf{b}$. Let us partition the nodes into three sets as follows:



$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix}$$

Now suppose that we can somehow construct \mathbf{A}_{11}^{-1} and \mathbf{A}_{22}^{-1} . Then

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{A}_{31}\mathbf{A}_{11}^{-1} & \mathbf{A}_{32}\mathbf{A}_{22}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{A}_{11}^{-1}\mathbf{A}_{13} \\ \mathbf{0} & \mathbf{I} & \mathbf{A}_{22}^{-1}\mathbf{A}_{23} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement*.

In other words, in order to invert \mathbf{A} , we need to execute three steps:

- Invert \mathbf{A}_{11} to form \mathbf{A}_{11}^{-1} .
- Invert \mathbf{A}_{22} to form \mathbf{A}_{22}^{-1} .
- Invert $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$.

size $\sim N/2 \times N/2$

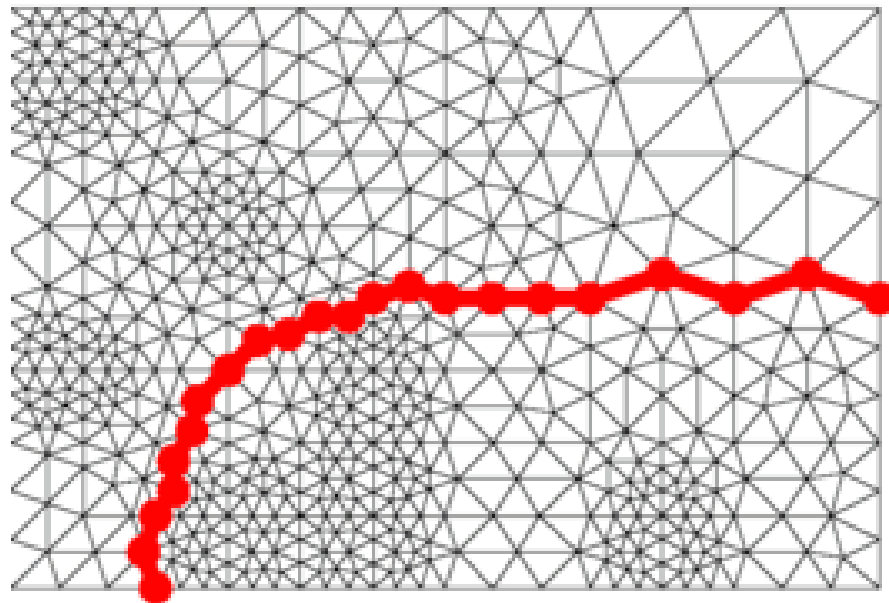
size $\sim N/2 \times N/2$

size $\sim \sqrt{N} \times \sqrt{N}$

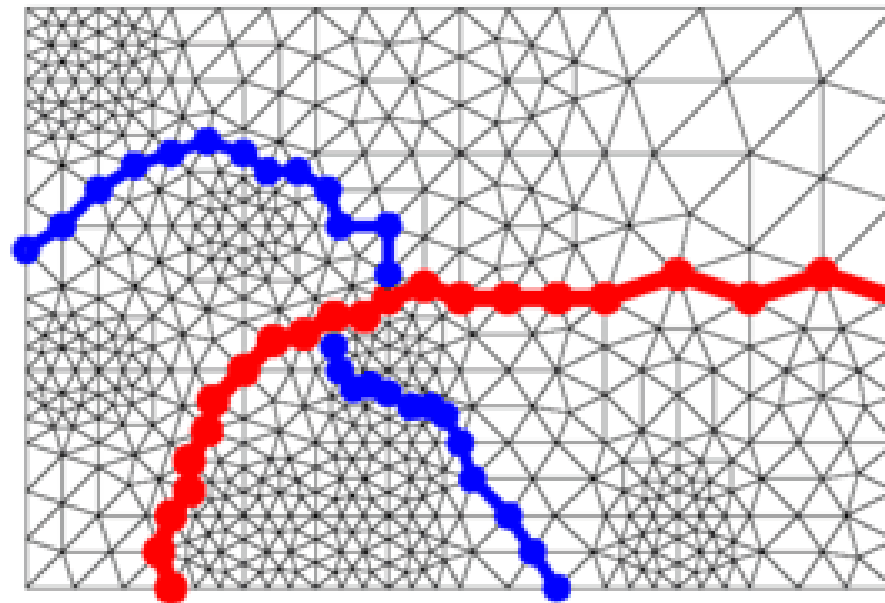
Notice the obvious recursion!

Example: Schur complements

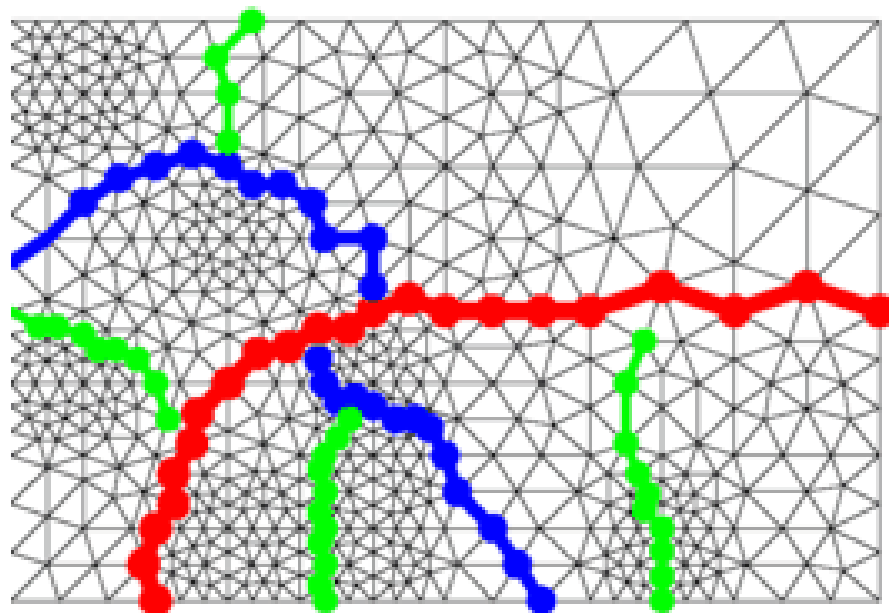
Typically, nested dissection orderings are more complicated:



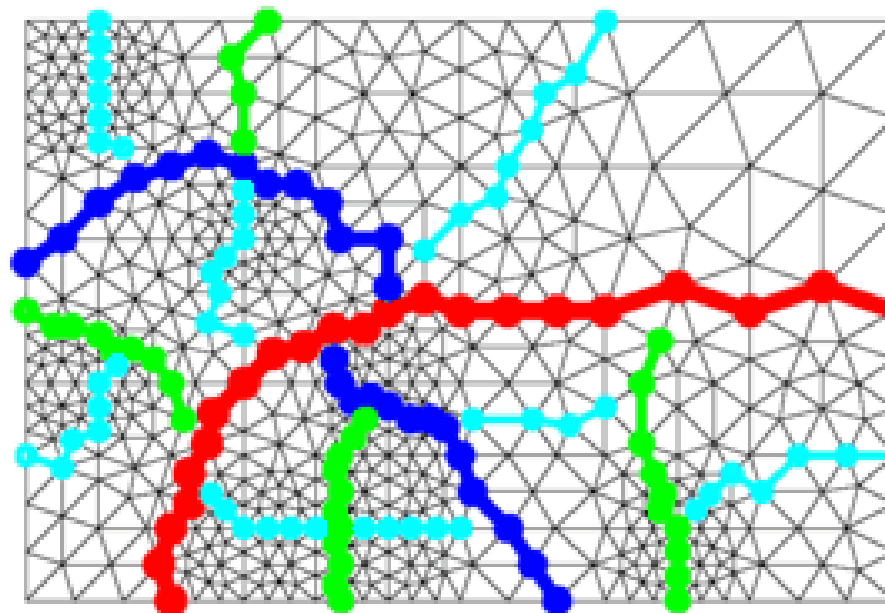
(i) *Top level separator*



(ii) *Two levels of separators*



(iii) *Three levels of separators*



(iv) *Four levels of separators*

Image credit: Jianlin Xia, "Robust and Efficient Multifrontal Solver for Large Discretized PDEs", 2012

Observe that while the computational domain is **2D** in this example, the rank structured matrices all live on the colored **1D** domains.

Example: Schur complements

It is well known that the dense factorization of the largest Schur complements is the dominant cost in sparse LU factorization.

For problems in 2D, the asymptotic flop count is $O(N^{1.5})$.

For problems in 3D, the asymptotic flop count is $O(N^2)$.

Example: Schur complements

It is well known that the dense factorization of the largest Schur complements is the dominant cost in sparse LU factorization.

For problems in 2D, the asymptotic flop count is $O(N^{1.5})$.

For problems in 3D, the asymptotic flop count is $O(N^2)$.

Assertion: These Schur complements very often behave like discretized integral operators. (E.g. Dirichlet-to-Neumann.)

They are rank-structured, and are amenable to “fast” matrix algebra. Exploiting this, the complexity of sparse direct solvers for elliptic PDEs has in the past 10 – 20 years been decreased dramatically:

	<i>Build stage</i>		<i>Solve stage</i>	
2D	$N^{3/2}$	$\rightarrow N$	$N \log N$	$\rightarrow N$
3D	N^2	$\rightarrow N$	$N^{4/3}$	$\rightarrow N$

Example: Schur complements

It is well known that the dense factorization of the largest Schur complements is the dominant cost in sparse LU factorization.

For problems in 2D, the asymptotic flop count is $O(N^{1.5})$.

For problems in 3D, the asymptotic flop count is $O(N^2)$.

Nested dissection solvers with $O(N)$ complexity — Le Borne, Grasedyck, & Kriemann (2007), Martinsson (2009), **J. Xia**, Chandrasekaran, Gu, & Li (2009), Gillman & Martinsson (2011), Schmitz & **L. Ying** (2012), Darve & Ambikasaran (2013), Ho & Ying (2015), Amestoy, Ashcraft, et al (2015), Oseledets & Suchnikova (2015), etc.

$O(N)$ direct solvers for integral equations were developed by Martinsson & Rokhlin (2005), Greengard, Gueyffier, Martinsson, & Rokhlin (2009), Gillman, Young, & Martinsson (2012), Ho & Greengard (2012), Ho & Ying (2015). Work in 1990's Y. Chen, P. Starr, **V. Rokhlin**, **L. Greengard**, **E. Michielssen**. Related to work on \mathcal{H} and \mathcal{H}^2 matrix methods (1998 and forwards) by Börm, Bebendorf, Hackbusch, Khoromskij, Sauter, etc.

Example: Schur complements

It is well known that the dense factorization of the largest Schur complements is the dominant cost in sparse LU factorization.

For problems in 2D, the asymptotic flop count is $O(N^{1.5})$.

For problems in 3D, the asymptotic flop count is $O(N^2)$.

Note: Complexity is not $O(N)$ if the nr. of “points-per-wavelength” is fixed as $N \rightarrow \infty$. This limits direct solvers to problems of size a couple hundreds of wave-lengths or so.

More complicated rank-structured formats — “butterfly matrices” — offer promise here, and initial results show great promise.

Interaction ranks: Why are they small?

We have claimed that a wide range of global operators that arise in scientific computing have rank structure. Specifically, the claim is that the numerical rank of interaction between two subdomains that are separated in space is low.

Why is this the case?

Interaction ranks: Why are they small?

We have claimed that a wide range of global operators that arise in scientific computing have rank structure. Specifically, the claim is that the numerical rank of interaction between two subdomains that are separated in space is low.

Why is this the case?

(One) Answer: It is a consequence of the *smoothing effect* of elliptic differential equations; it can be interpreted as a *loss of information*.

This effect has many well known physical consequences:

- Rapid convergence of *multipole expansions* when the region of sources is far away from the observation point.
- The *St Venant principle* in mechanics.
- The inaccuracy of imaging at sub-wavelength scales.
- The intractability of solving the heat equation backwards.

Caveat: High-frequency problems present difficulties — no loss of information for length-scales $> \lambda$. Extreme accuracy of optics, high-frequency imaging, *etc.*

Interaction ranks: Boundary integral equations

Let us consider two simple boundary integral equations on a boundary Γ :

The first is a reformulation of a Dirichlet problem involving the Laplace equation:

$$\alpha\sigma(\mathbf{x}) + \int_{\Gamma} (d(\mathbf{x}, \mathbf{y}) + s(\mathbf{x}, \mathbf{y})) \sigma(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

The second is a reformulation of a Dirichlet problem involving the Helmholtz equation:

$$\beta\sigma(\mathbf{x}) + \int_{\Gamma} (d_{\kappa}(\mathbf{x}, \mathbf{y}) + i\kappa s_{\kappa}(\mathbf{x}, \mathbf{y})) \sigma(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

The kernels are derived from the corresponding fundamental solutions:

$$s(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x} - \mathbf{y}),$$

$$d(\mathbf{x}, \mathbf{y}) = \partial_{\mathbf{n}(\mathbf{y})} \phi(\mathbf{x} - \mathbf{y}),$$

$$s_{\kappa}(\mathbf{x}, \mathbf{y}) = \phi_{\kappa}(\mathbf{x} - \mathbf{y}),$$

$$d_{\kappa}(\mathbf{x}, \mathbf{y}) = \partial_{\mathbf{n}(\mathbf{y})} \phi_{\kappa}(\mathbf{x} - \mathbf{y}),$$

where, as before,

$$\phi(\mathbf{x}) = -\frac{1}{2\pi} \log |\mathbf{x}|,$$
$$\phi_{\kappa}(\mathbf{x}) = \frac{i}{4} H_0^{(1)}(\kappa |\mathbf{x}|).$$

Interaction ranks: Boundary integral equations

Let us consider two simple boundary integral equations on a boundary Γ :

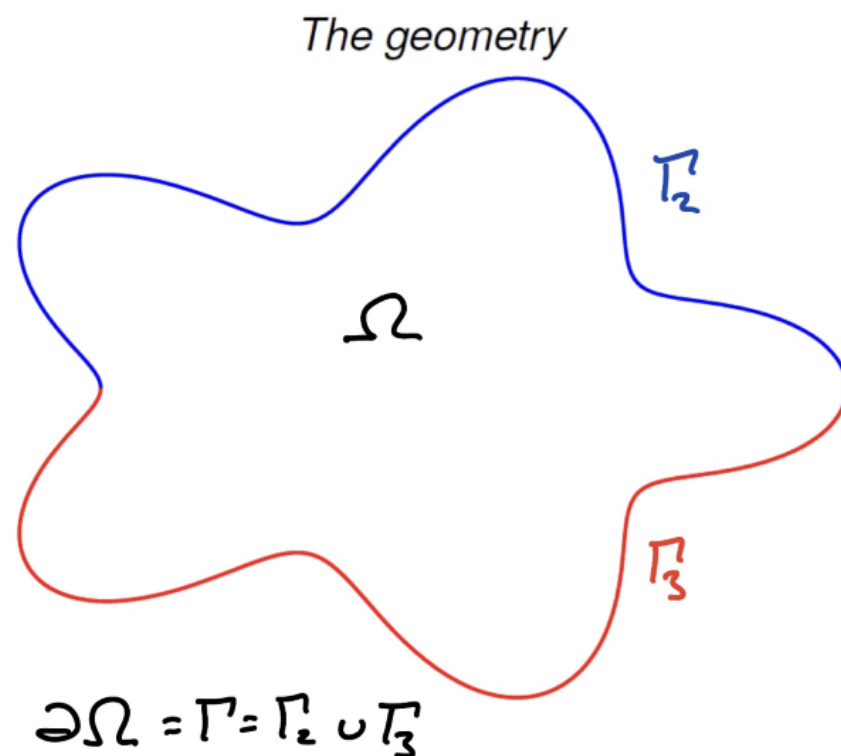
The first is a reformulation of a Dirichlet problem involving the Laplace equation:

$$\alpha\sigma(\mathbf{x}) + \int_{\Gamma} (d(\mathbf{x}, \mathbf{y}) + s(\mathbf{x}, \mathbf{y})) \sigma(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

The second is a reformulation of a Dirichlet problem involving the Helmholtz equation:

$$\beta\sigma(\mathbf{x}) + \int_{\Gamma} (d_{\kappa}(\mathbf{x}, \mathbf{y}) + i\kappa s_{\kappa}(\mathbf{x}, \mathbf{y})) \sigma(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

Let \mathbf{A} denote the matrix resulting from discretization of either BIE.



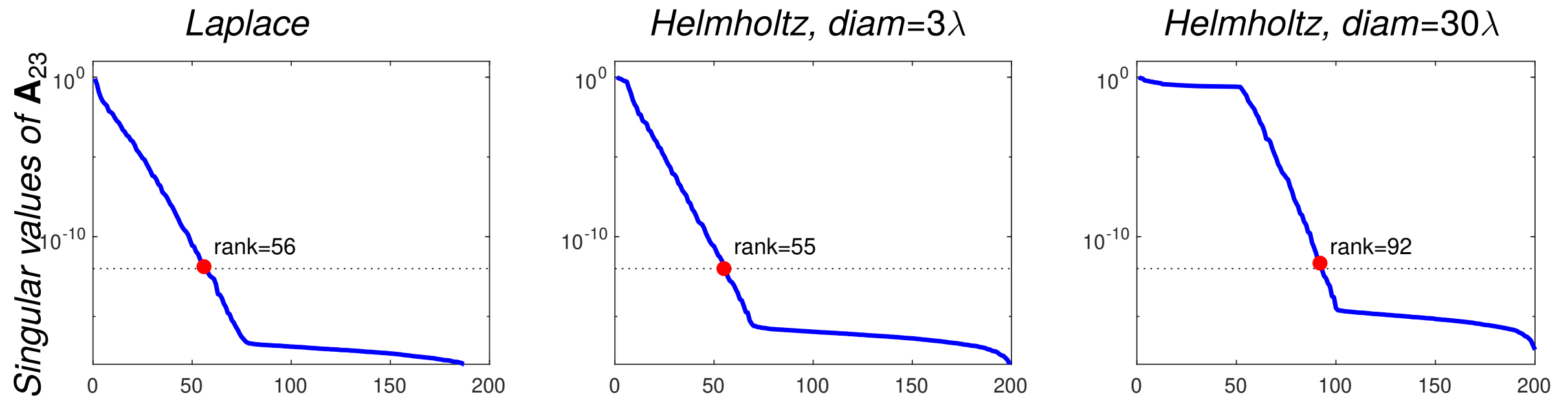
The matrix \mathbf{A}

A_{22}	A_{23}
A_{32}	A_{33}

On the next slide, we show the singular values of the off-diagonal block \mathbf{A}_{23} .

Interaction ranks: Boundary integral equations

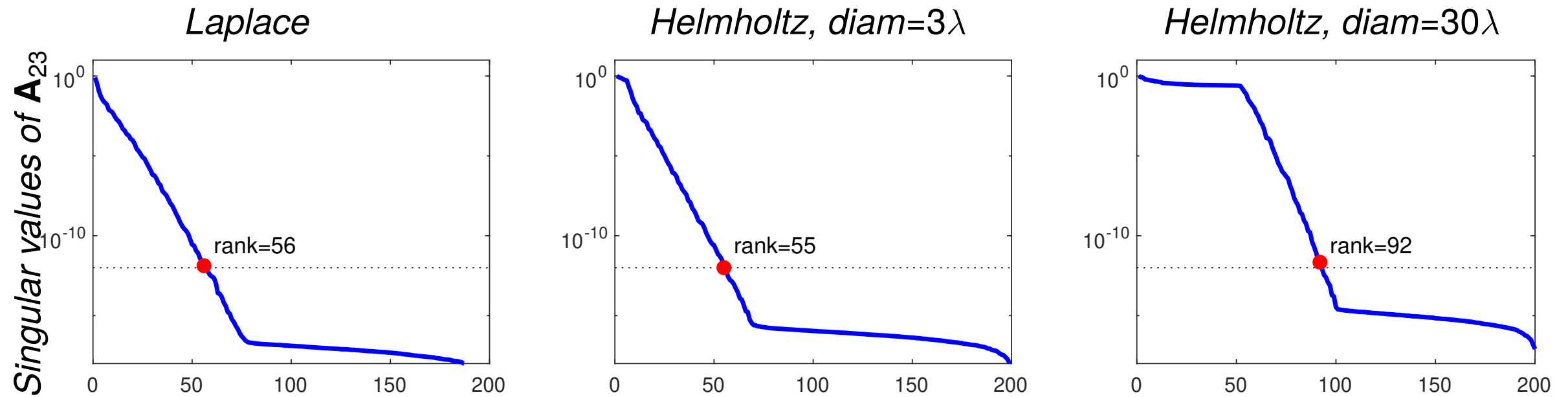
The ranks of an off-diagonal block of \mathbf{A} :



This is all as expected. Somewhat accessible by analysis.

Interaction ranks: Boundary integral equations

The ranks of an off-diagonal block of \mathbf{A} :

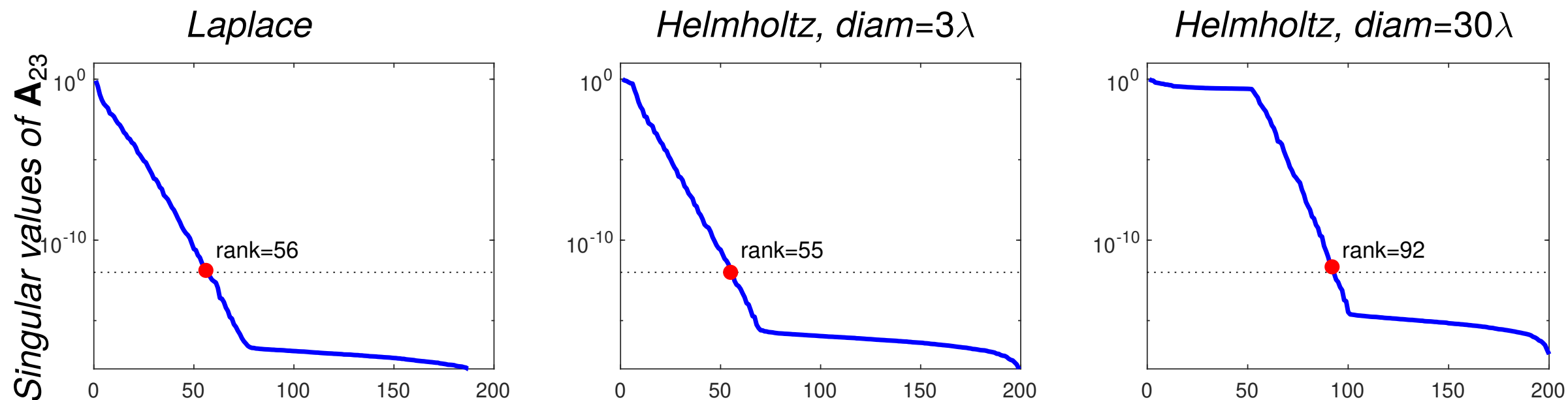


This is all as expected. Somewhat accessible by analysis.

Now the fun part! We set $\mathbf{B} = \mathbf{A}^{-1}$, and plot the svds of the off-diagonal block \mathbf{B}_{23} .

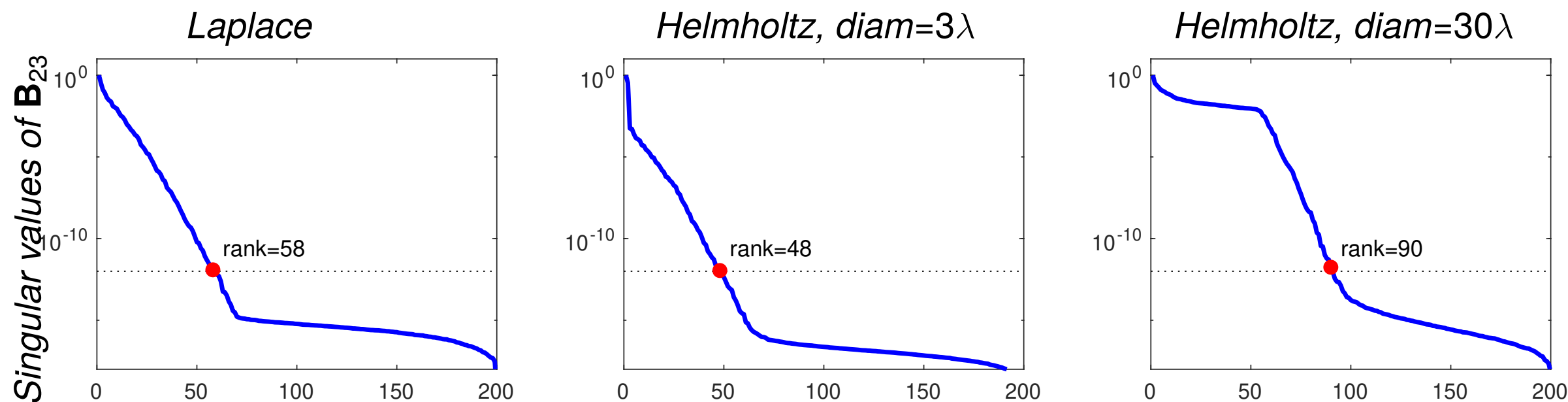
Interaction ranks: Boundary integral equations

The ranks of an off-diagonal block of \mathbf{A} :



This is all as expected. Somewhat accessible by analysis.

Now the fun part! We set $\mathbf{B} = \mathbf{A}^{-1}$, and plot the svds of the off-diagonal block \mathbf{B}_{23} .

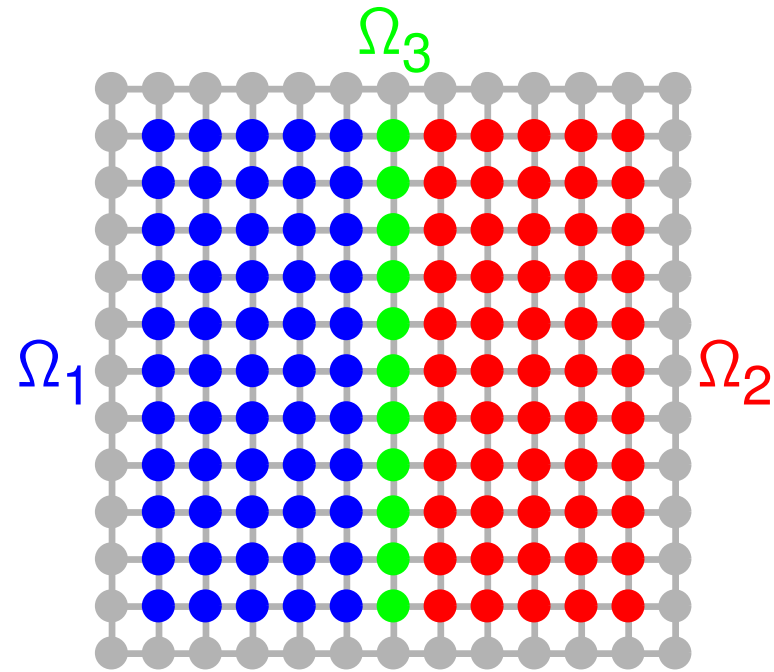


Remarkable similarity!

(Observe ill-conditioning due to close resonances for the Helmholtz BIE.)

Interaction ranks: Stiffness matrix from finite difference discretization

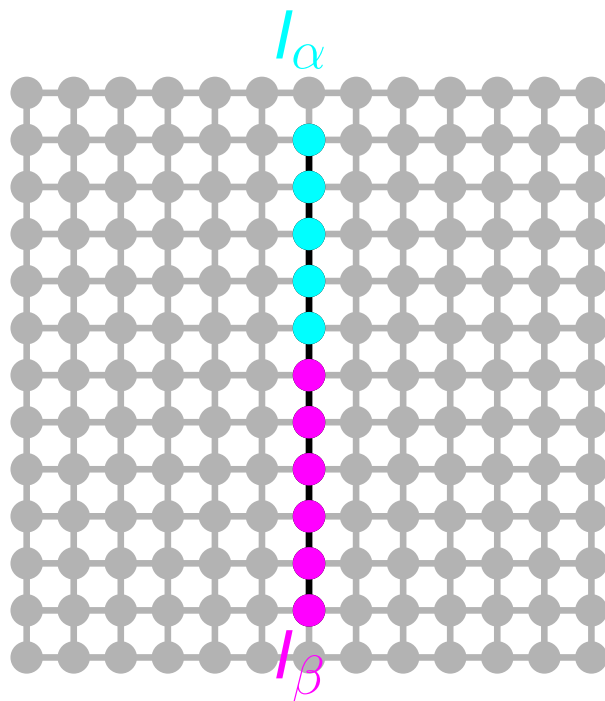
Recall our example of Laplace's equation discretized using the 5-point stencil.



$$\mathbf{A} = \begin{array}{|c|c|c|} \hline \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \\ \hline \end{array}$$

We build the Schur complement $\mathbf{S} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$.

Then split the Schur complement into four parts:

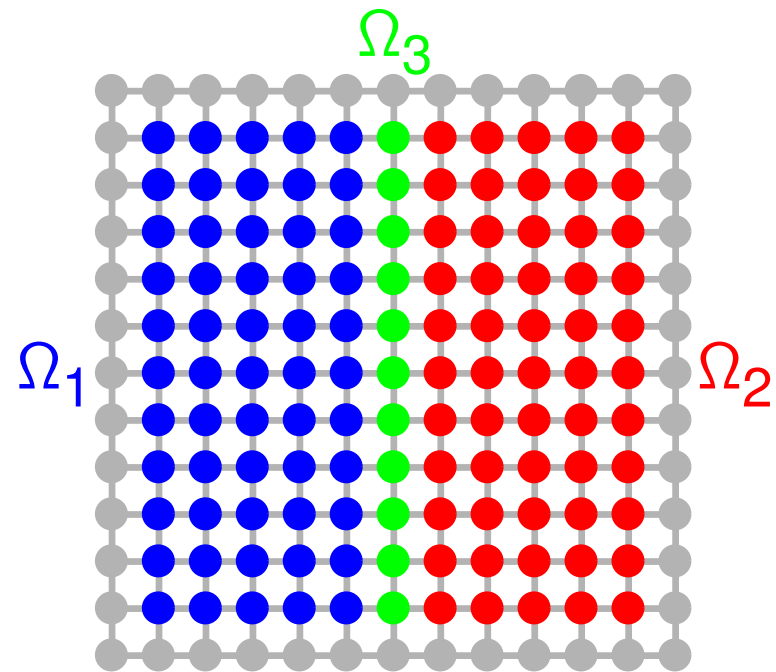


$$\mathbf{S} = \begin{array}{|c|c|} \hline \mathbf{S}_{\alpha\alpha} & \mathbf{S}_{\alpha\beta} \\ \hline \mathbf{S}_{\beta\alpha} & \mathbf{S}_{\beta\beta} \\ \hline \end{array}$$

We explore the svds of $\mathbf{S}_{\alpha\beta}$ — encoding interactions between I_α and I_β .

Interaction ranks: Stiffness matrix from finite difference discretization

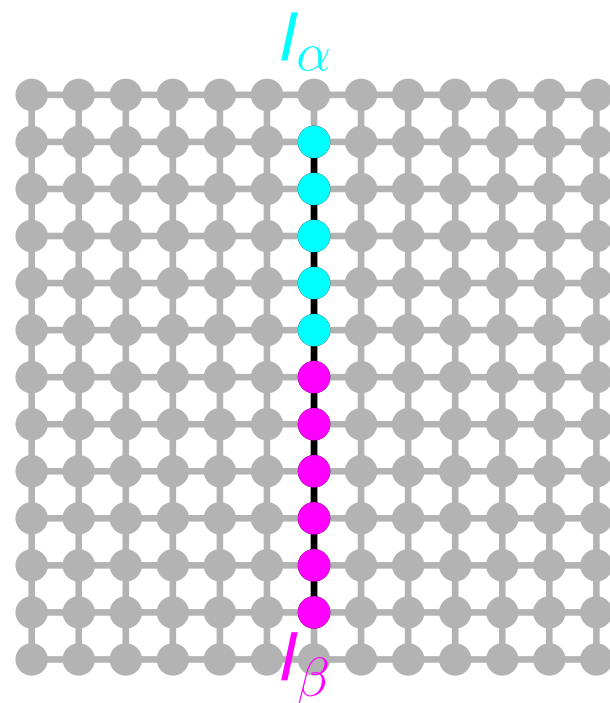
Recall our example of Laplace's equation discretized using the 5-point stencil.



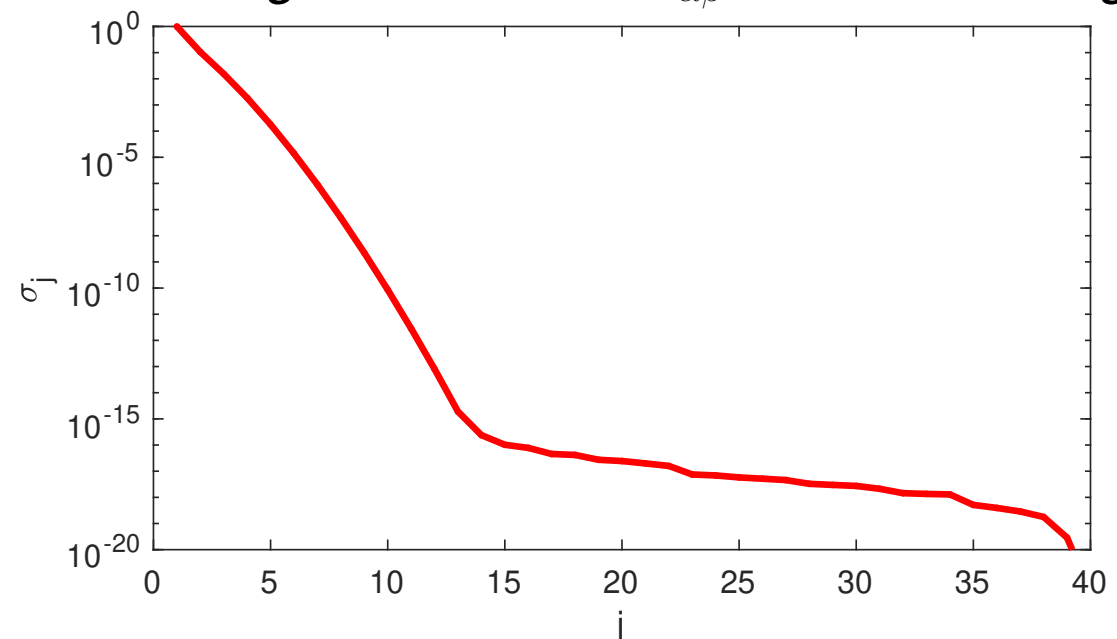
$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix}$$

We build the Schur complement $\mathbf{S} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$.

Then split the Schur complement into four parts:



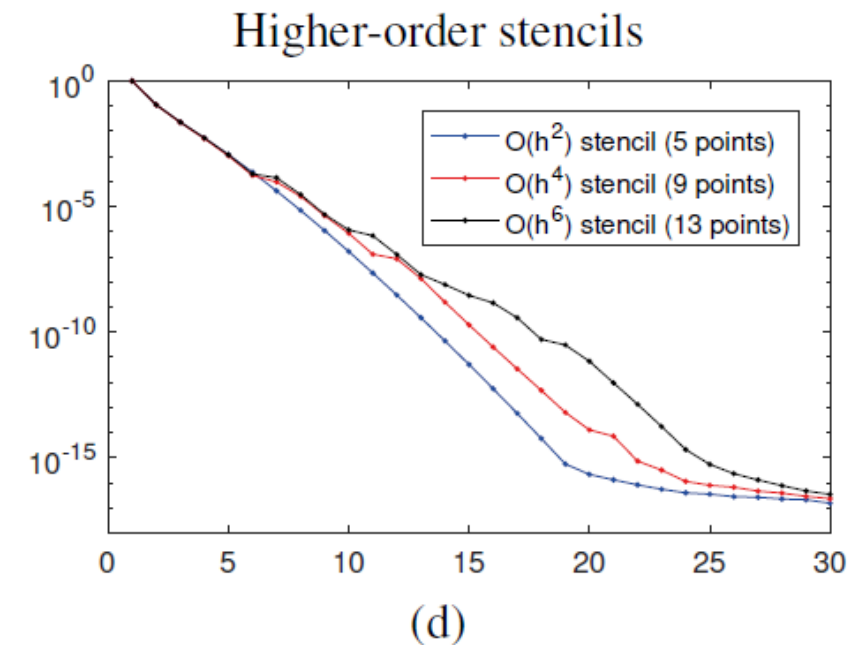
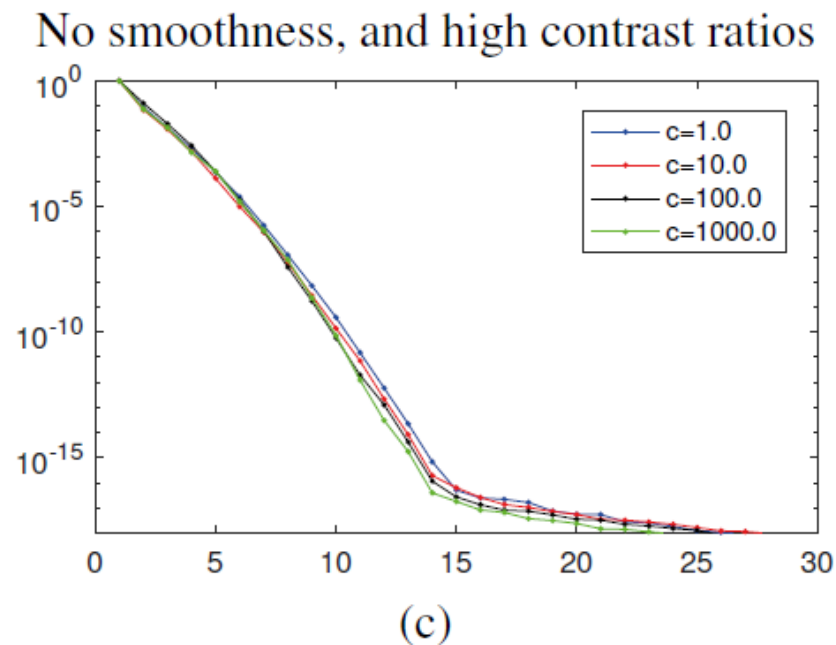
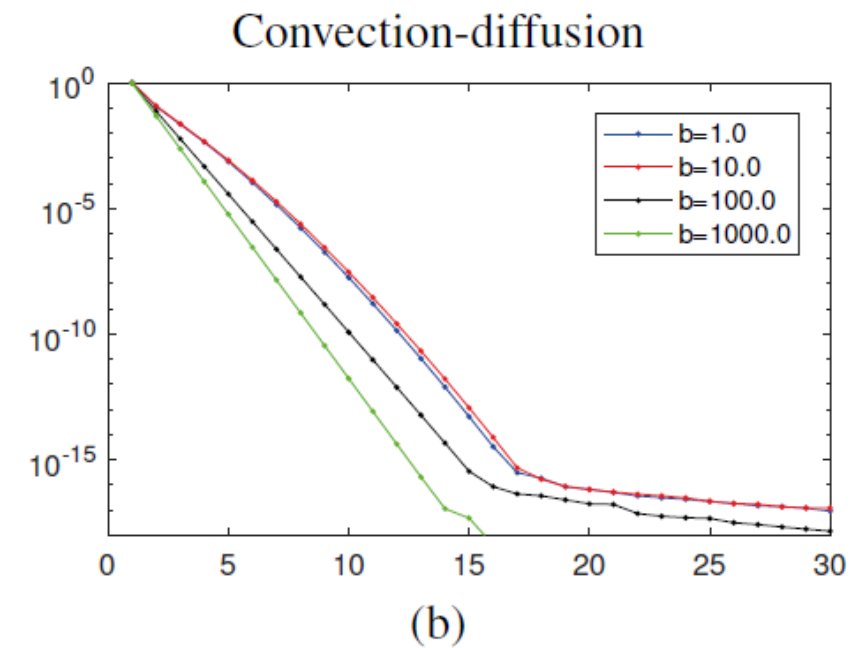
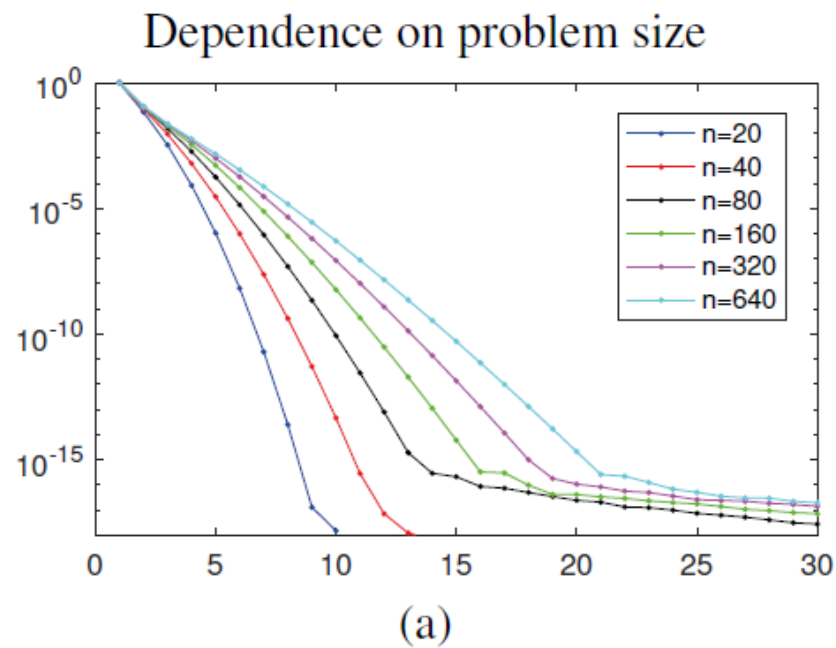
Singular values of $S_{\alpha\beta}$ for an 80×80 grid.



We explore the svds of $\mathbf{S}_{\alpha\beta}$ — encoding interactions between I_α and I_β .

Interaction ranks: Stiffness matrix from finite difference discretization

Let us try a few different PDEs, and different problem sizes:



Note: The rank decay property is remarkably stable!

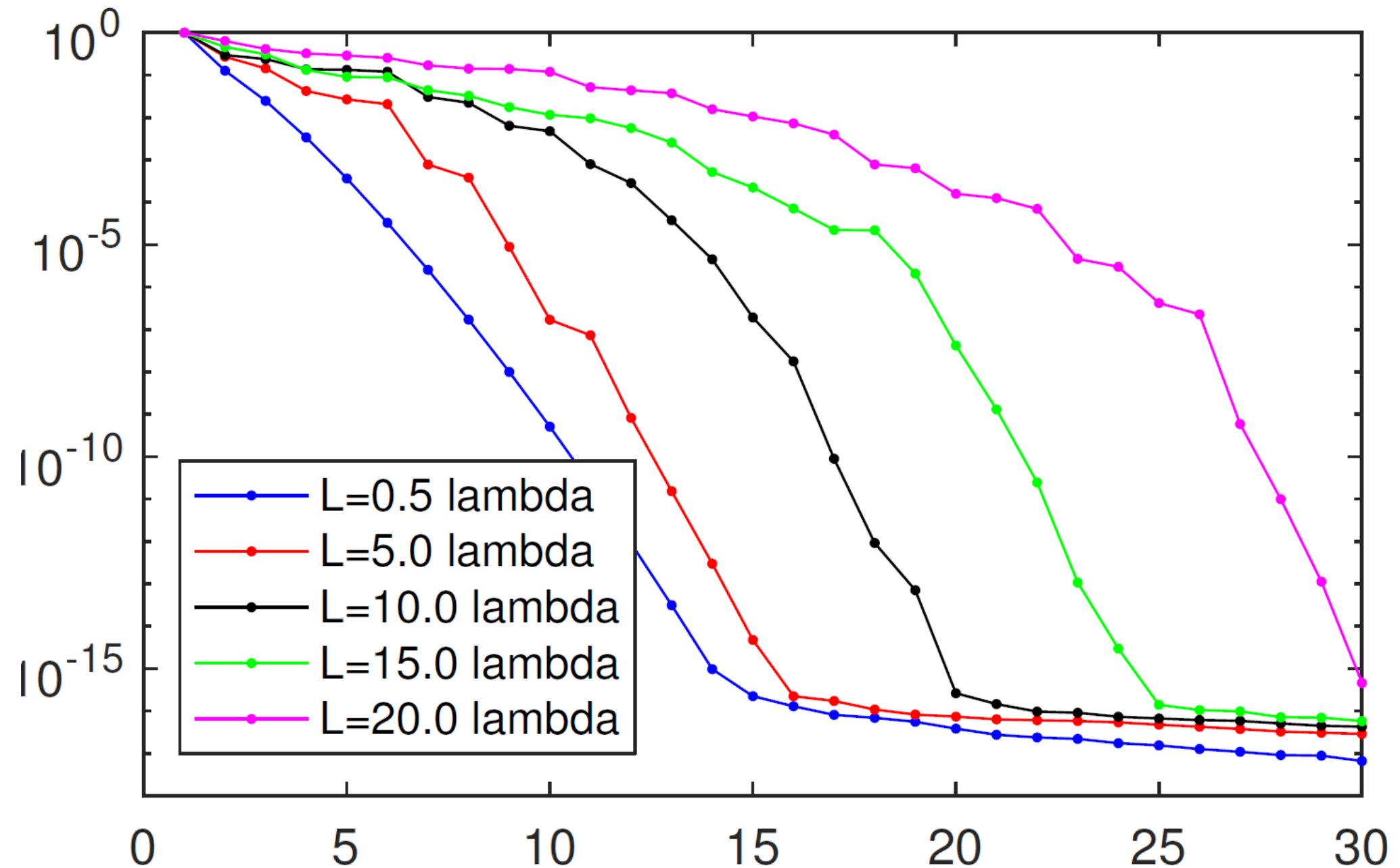
Note: The decay continues to ϵ_{mach} — regardless of the discretization errors!

Interaction ranks: Stiffness matrix from finite difference discretization

Next, let us consider Helmholtz problems with increasing wave numbers.

Interaction ranks: Stiffness matrix from finite difference discretization

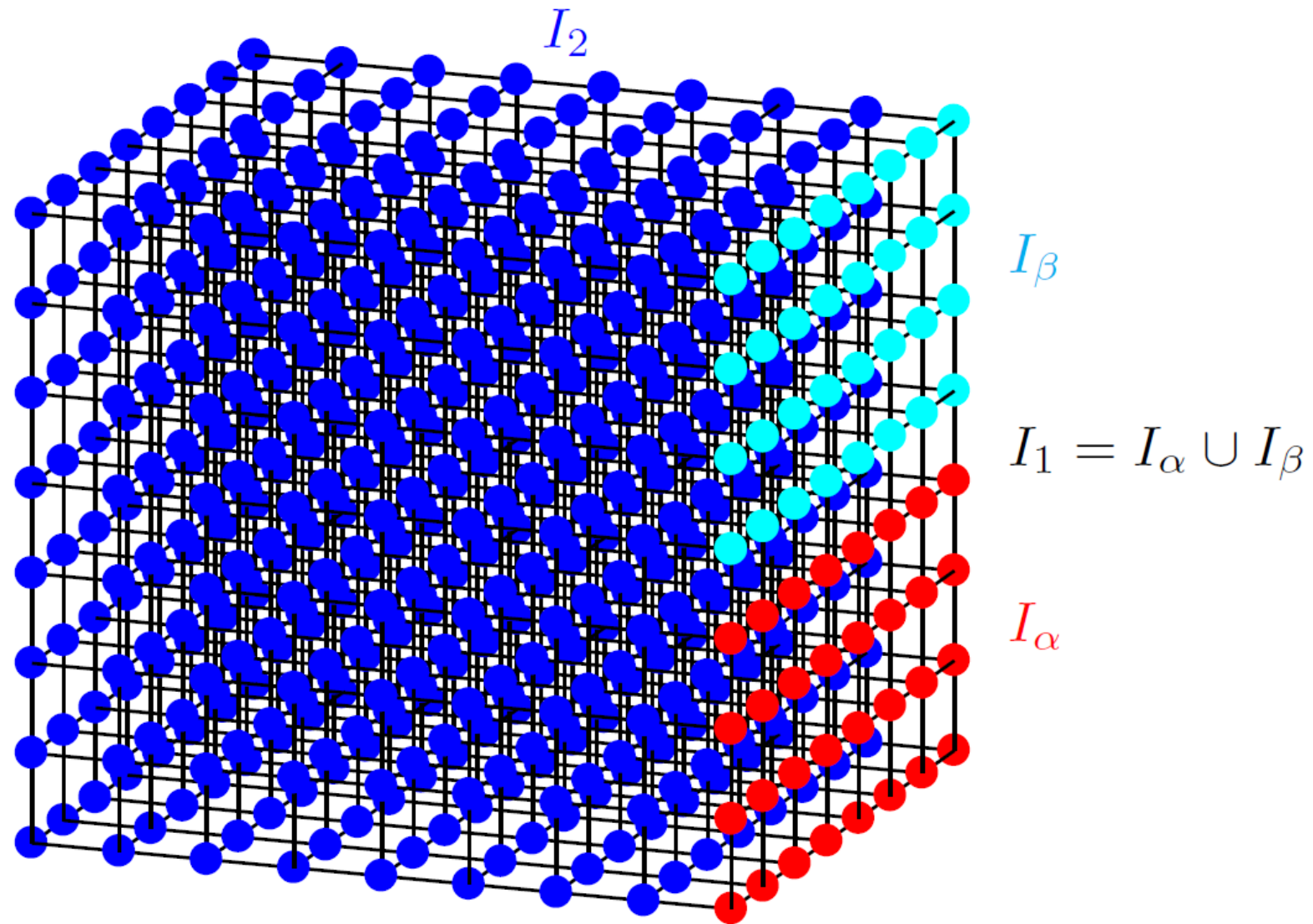
Next, let us consider Helmholtz problems with increasing wave numbers.



Fast decay once oscillations are resolved.

Interaction ranks: Stiffness matrix from finite difference discretization

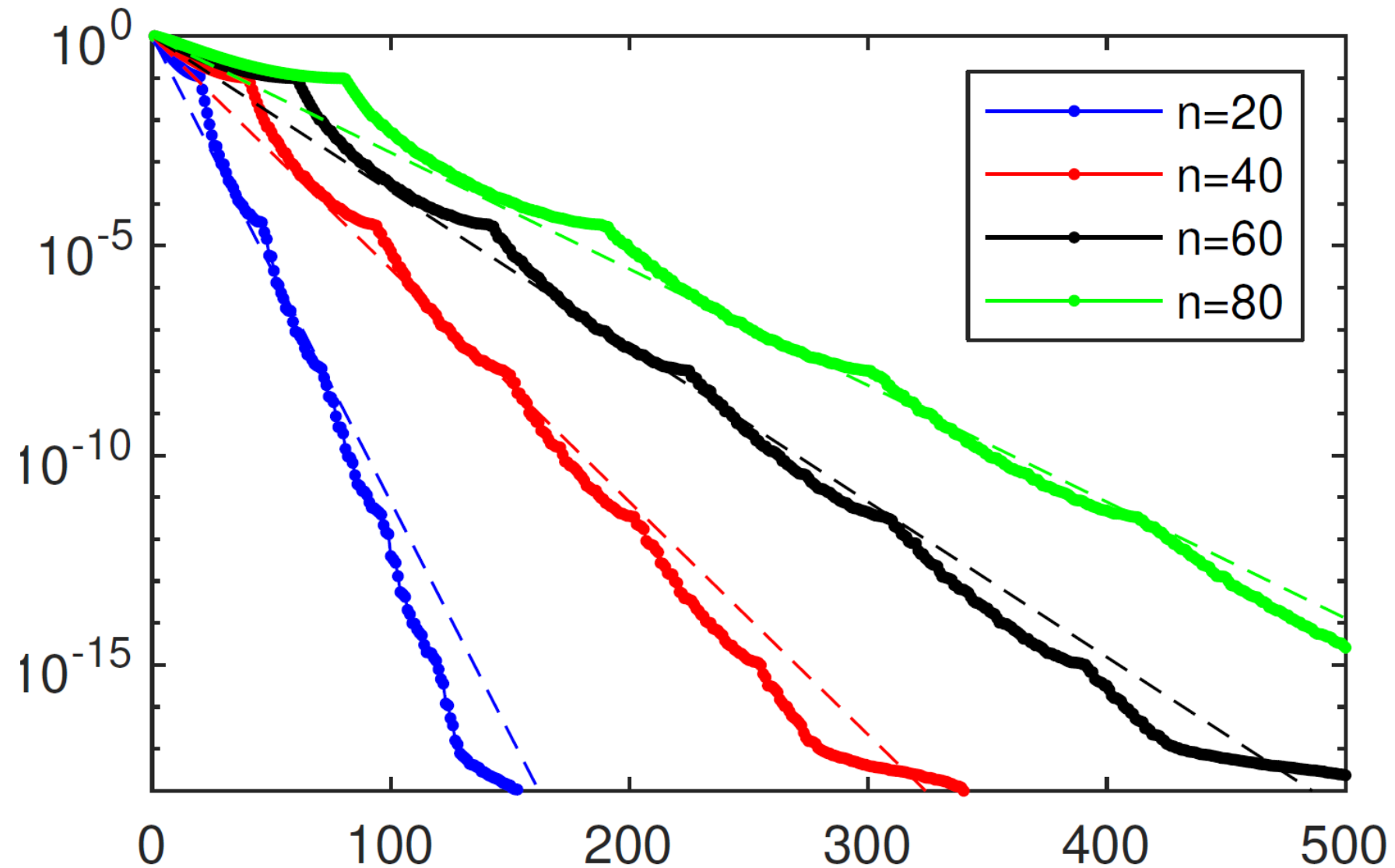
Finally, let us consider the analogous 3D problem.



The geometry.

Interaction ranks: Stiffness matrix from finite difference discretization

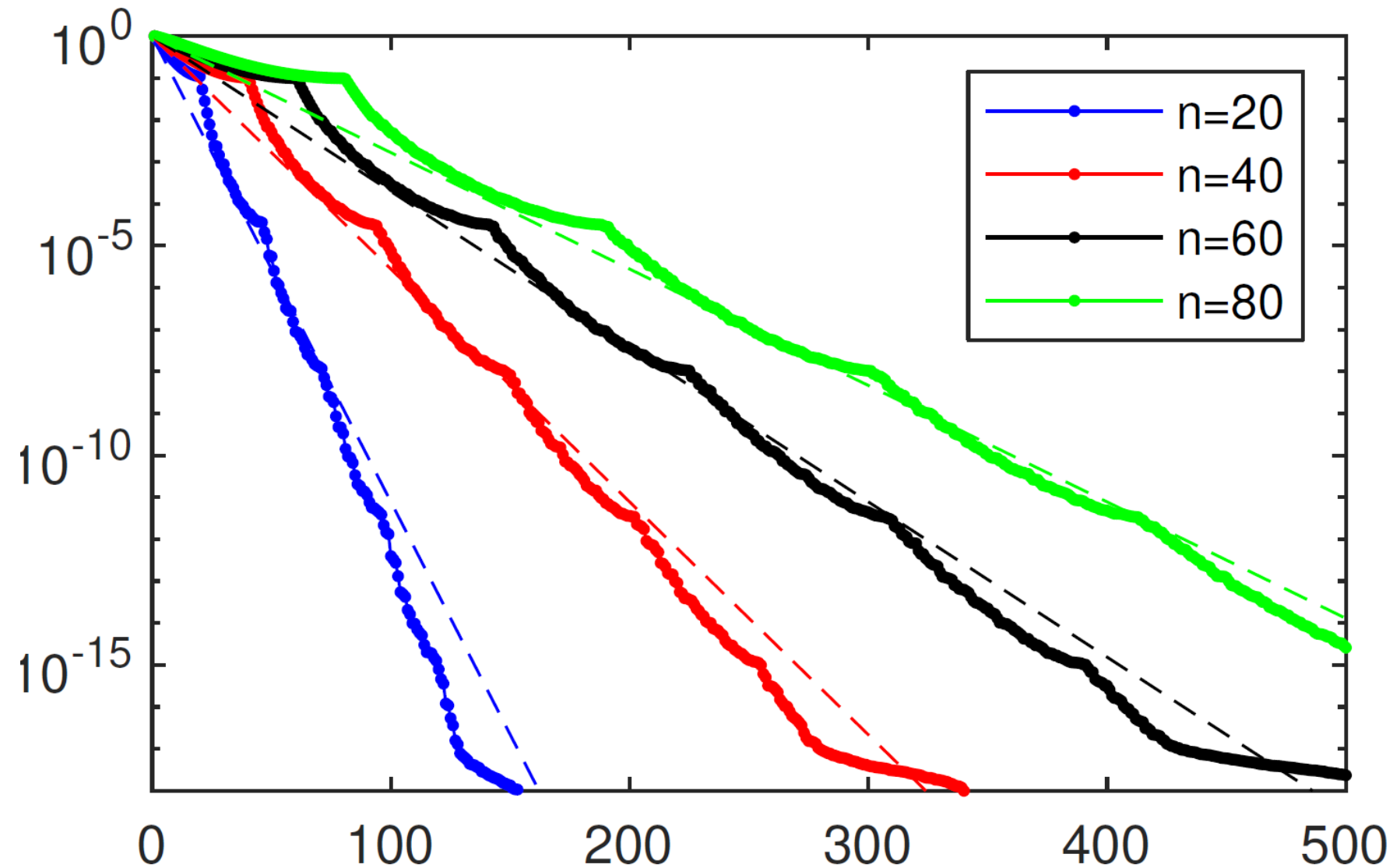
Finally, let us consider the analogous 3D problem.



The singular values.

Interaction ranks: Stiffness matrix from finite difference discretization

Finally, let us consider the analogous 3D problem.



The singular values.

If you make similar plots for Dirichlet-to-Neumann operators, or other Poincaré-Steklov operators, the general behavior will be the same.

Compression of a matrix discretizing a continuum operator

Question: How do you obtain the data sparse representation of a dense matrix discretizing a continuum operator?

Compression of a matrix discretizing a continuum operator

Question: How do you obtain the data sparse representation of a dense matrix discretizing a continuum operator?

- In some standard environments (convolution with a known fundamental solution, say), there exist analytic techniques that work very well. “Abramowitz & Stegun” or “proxy surface method”. *Classical FMM environment*
- In some cases where the kernel is known explicitly, heuristic techniques such as “adaptive cross approximation” are often used. These are fast, but not 100% reliable. Ok for building pre-conditioners. *H-matrix environment*

Compression of a matrix discretizing a continuum operator

Question: How do you obtain the data sparse representation of a dense matrix discretizing a continuum operator?

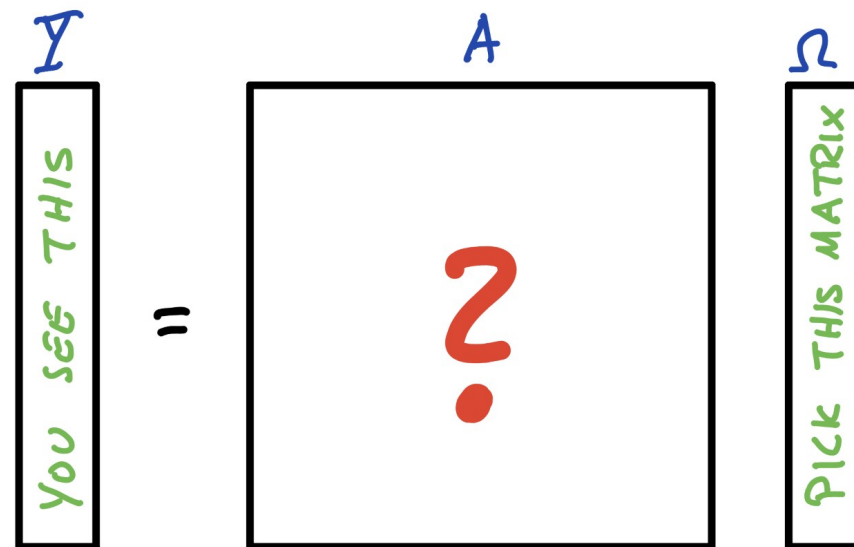
- In some standard environments (convolution with a known fundamental solution, say), there exist analytic techniques that work very well. “Abramowitz & Stegun” or “proxy surface method”. *Classical FMM environment*
- In some cases where the kernel is known explicitly, heuristic techniques such as “adaptive cross approximation” are often used. These are fast, but not 100% reliable. Ok for building pre-conditioners. *H-matrix environment*
- In more general cases, *randomized* algorithms are very competitive. These methods require that you have some means of applying the operator (e.g. a legacy PDE solver), so that you can observe input-output pairs.

Compression of a rank-structured matrix

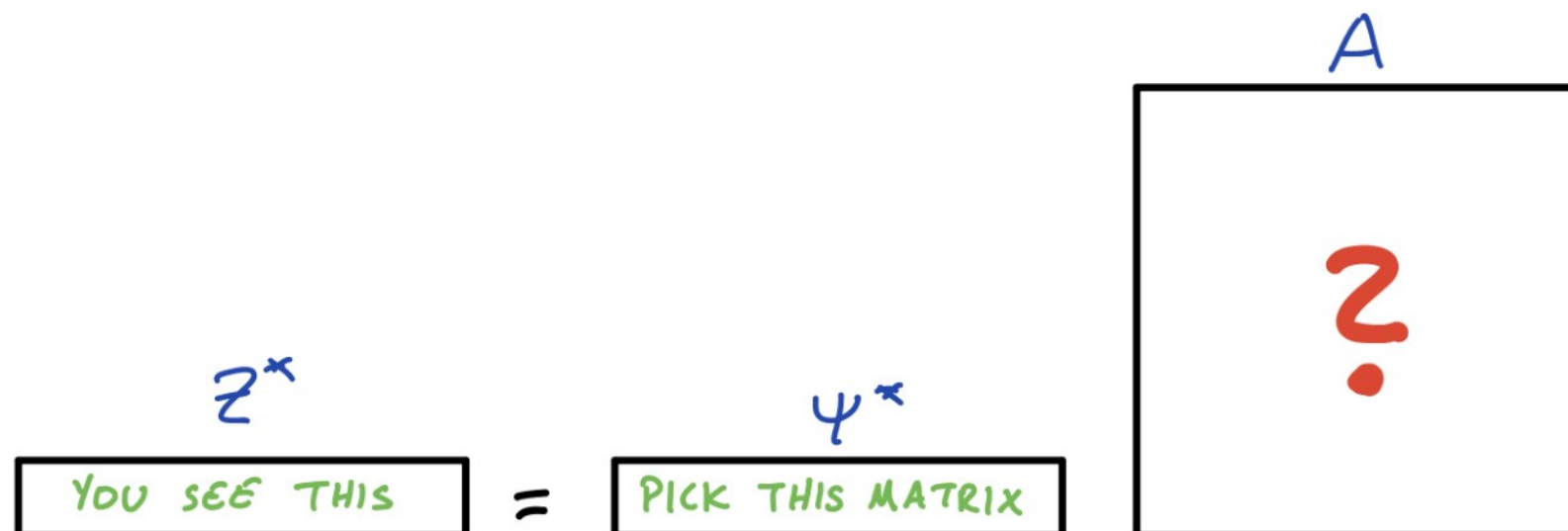
Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

Objective: Construct thin matrices Ω and Ψ such that \mathbf{A} can be completely reconstructed in $O(N)$ work from the set $\{\mathbf{Y}, \Omega, \mathbf{Z}, \Psi\}$ where $\mathbf{Y} = \mathbf{A}\Omega$ and $\mathbf{Z} = \mathbf{A}^*\Psi$?

Sample the column space of the matrix:



If $\mathbf{A} \neq \mathbf{A}^$, then sample the row space too:*



Compression of a rank-structured matrix

Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

Objective: Construct thin matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ such that \mathbf{A} can be completely reconstructed in $O(N)$ work from the set $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$ where $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$?

The low rank case: In the particularly simple case where \mathbf{A} has *global* rank k , we revert to the case we considered in the first part of the talk.

In the current framework, the randomized SVD takes the form:

- Set $s = k$ and draw a “test matrix” $\mathbf{\Omega} \in \mathbb{R}^{N \times s}$ from a Gaussian distribution.
- Form the “sample matrix” $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.
- Build $\mathbf{\Psi}$ to hold an ON basis for $\text{ran}(\mathbf{Y})$, e.g., $[\mathbf{\Psi}, \sim] = \text{qr}(\mathbf{Y}, 0)$.
- Form $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$.

Then $\mathbf{A} = \mathbf{\Psi} (\mathbf{\Psi}^* \mathbf{A}) = \mathbf{\Psi} \mathbf{Z}^*$ with probability 1.

In the more typical case where \mathbf{A} is only *approximately* of rank k , some *oversampling* is required to get a reliable scheme. (Say $s = k + 10$, or $s = 2k$, or some such.)

Rank structured case: Extract *all* the low-rank matrices, and *all* the dense blocks, from a very limited set of global “probes”. How do you disentangle the mixed samples?

Compression of a rank-structured matrix

Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

Objective: Construct thin matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ such that \mathbf{A} can be completely reconstructed in $O(N)$ work from the set $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$ where $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$?

Why generalize from “global low rank” to “rank-structured”:

- Integral operators from classical physics. If you have a legacy method for the matrix-vector multiple (e.g. the Fast Multipole Method), then we could enable a range of operations – LU factorization, matrix inversion, etc.
- Compute Dirichlet-to-Neumann (or Impedance-to-Impedance) operators explicitly whenever you have access to a fast PDE solvers.
- Compression of Schur complements that arise in the LU or Cholesky factorization of sparse matrices. This overcomes the key computational bottleneck, and for instance admits the acceleration of the LU factorization of a “finite element” matrix *from $O(N^2)$ to near linear complexity.*
- Multiplication of operators. \rightarrow Multiphysics, multi-modal discretizations, etc.

Compression of a rank-structured matrix

Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

Objective: Construct thin matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ such that \mathbf{A} can be completely reconstructed in $O(N)$ work from the set $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$ where $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$?

Available techniques for the rank structured case:

For the most general structured matrix formats (e.g. \mathcal{H} -matrices), the problem has been solved in principle, and close to linear complexity algorithms exist:

- L. Lin, J. Lu, L. Ying, JCP, **230**(10), pp. 4071–4087, 2011.
- P.G. Martinsson, SISC, **38**(4), pp. A1959-A1986, 2016.

However, existing methods require $\sim k \log(N)$ matvecs, and do not have great practical speed. For instance, as dimension d increases, the bound on flops has an 8^d factor ...

Recently proposed algorithms have reduced the pre-factors by constructing bespoke random matrices that are designed to be optimal for any given tessellation pattern. The key technical idea is to formulate admissibility criteria that form a graph, and then exploit powerful graph coloring algorithms. This technique also enables compression of kernel matrices that arise in ML. [*J. Levitt & P.G. Martinsson, JCAM 451(1), 2024.*]

Compression of a rank-structured matrix

Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

Objective: Construct thin matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ such that \mathbf{A} can be completely reconstructed in $O(N)$ work from the set $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$ where $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$?

Available techniques for the rank structured case:

Related work:

Randomized compression of butterfly matrices: Path towards medium/high frequency problems. [Y. Liu, X. Xing, H. Guo, E. Michielssen, P. Ghysels, X.S. Li. 2021], [Y. Li, H. Yang, E. Martin, K. Ho, and L. Ying, 2015].

Tagging of random matrices: Highly efficient technique for building bases for off-diagonal blocks in the strong admissibility case. [K. Pearce, A. Yesypenko, J. Levitt, P.G. Martinsson arXiv:2501.05528]

Randomized strong recursive skeletonization: Do inversion and compression simultaneously \rightarrow order of magnitude reduction in memory requirements. [Anna Yesypenko PhD thesis, UT-Austin, Nov. 2023. Arxiv:2311.01451. To appear in J. Sci. Comp.]

Complexity analysis – what is the minimal number of matvecs required? Recent work by A. Townsend, D. Halikias, C. Musco & C. Musco, D. Persson, N. Boullé.

Compression of a rank-structured matrix

Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

Objective: Construct thin matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ such that \mathbf{A} can be completely reconstructed in $O(N)$ work from the set $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$ where $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$?

Available techniques for the rank structured case:

Good news is that in the context of *numerical PDEs*, more specialized rank structured formats are often sufficient — hierarchically semi-separable matrices, hierarchically block-separable matrices, “ \mathcal{H} -matrices with weak admissibility”, etc.

For these matrices, algorithms with true linear complexity and high practical speed exist.

First generation algorithms were not fully black box, as they required the ability to evaluate a small number of matrix entries explicitly.

- P.G. Martinsson, SIMAX, **32**(4), 2011.
- Later improvements by Jianlin Xia, Sherry Li, and others. Widely used.

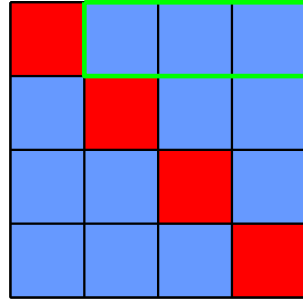
Recent: Fully black box algorithm with true linear complexity and high practical speed:

- J. Levitt & P.G. Martinsson, SISC, **46**(3), 2024.

Compression of a rank-structured matrix – A naive approach

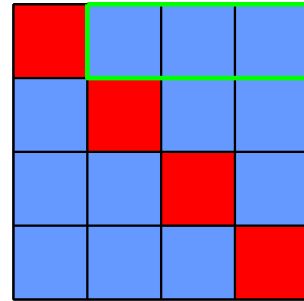
Consider the task of finding a basis matrix \mathbf{U}_4 for node 4 using randomized sampling.

We seek a sample of $\mathbf{A}(I_4, I_4^C)$, the HBS row block of node 4.

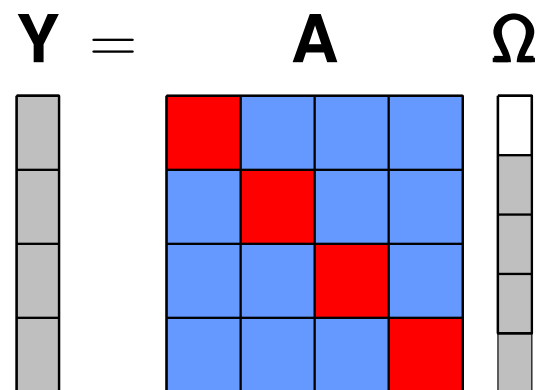


Compression of a rank-structured matrix – A naive approach

Consider the task of finding a basis matrix \mathbf{U}_4 for node 4 using randomized sampling. We seek a sample of $\mathbf{A}(I_4, I_4^C)$, the HBS row block of node 4.



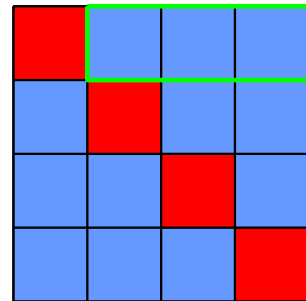
The naive approach is to sample with a random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4 . Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^C)$.



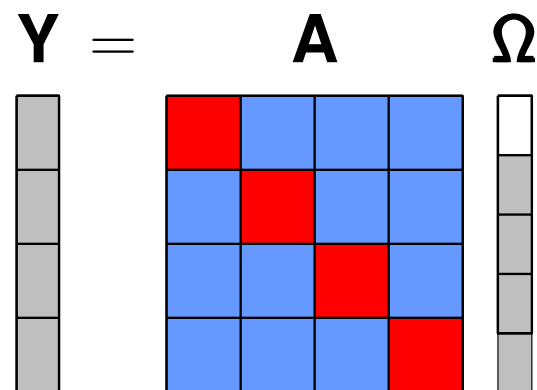
Compression of a rank-structured matrix – A naive approach

Consider the task of finding a basis matrix \mathbf{U}_4 for node 4 using randomized sampling.

We seek a sample of $\mathbf{A}(I_4, I_4^C)$, the HBS row block of node 4.



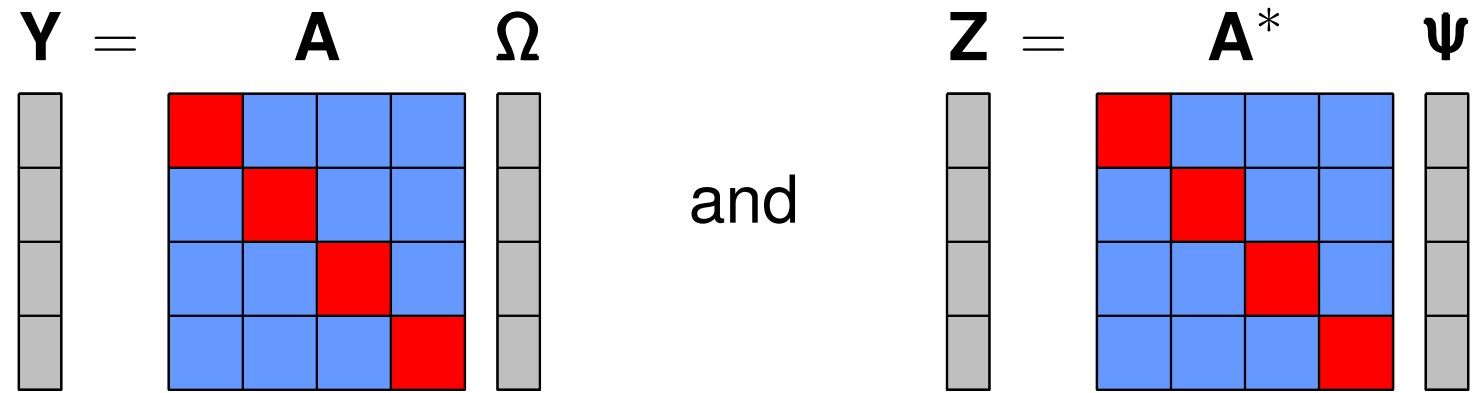
The naive approach is to sample with a random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4 . Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^C)$.



This scheme requires taking a separate set of r samples *for each leaf node*, for a total of $\sim rN/m$ samples. There is a lot of wasted information in \mathbf{Y} .

Compression of a rank-structured matrix – the “almost” black-box case

Sample \mathbf{A} with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:



Assumption: You can do matvecs and *entry evaluation*.

(More general soon.)

Compression of a rank-structured matrix – the “almost” black-box case

Sample \mathbf{A} with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:

$$\mathbf{Y} = \mathbf{A} \mathbf{\Omega} \quad \text{and} \quad \mathbf{Z} = \mathbf{A}^* \mathbf{\Psi}$$

Assumption: You can do matvecs and *entry evaluation*. (More general soon.)

In this case, we can explicitly form the diagonal blocks, and subtract their contributions:

$$\mathbf{Y}' = \mathbf{Y} - \mathbf{D} \mathbf{\Omega} = (\mathbf{A} - \mathbf{D}) \mathbf{\Omega}$$

Processing \mathbf{Z} analogously, we obtain basis matrices \mathbf{Y}_j and \mathbf{Z}_j for $j \in \{4, 5, 6, 7\}$ such that

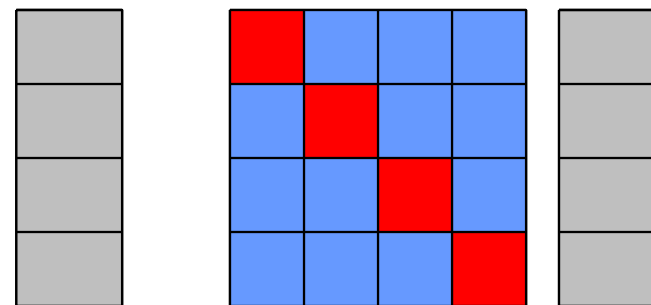
$$\mathbf{A}_{i,j} \approx \mathbf{Y}_i \mathbf{B}_{i,j} \mathbf{Z}_j^*, \quad i \neq j,$$

for *some* small matrices $\mathbf{B}_{i,j}$.

In a final step, use the ID to build the matrices $\mathbf{B}_{i,j}$ via entry evaluation.

Compression of a rank-structured matrix – fully black box

Sample \mathbf{A} with a completely dense random matrix $\Omega \in \mathbb{R}^{N \times (r+m)}$, where m is the leaf node size. (Think $m \approx 2k$ and $r = k + 10$.)

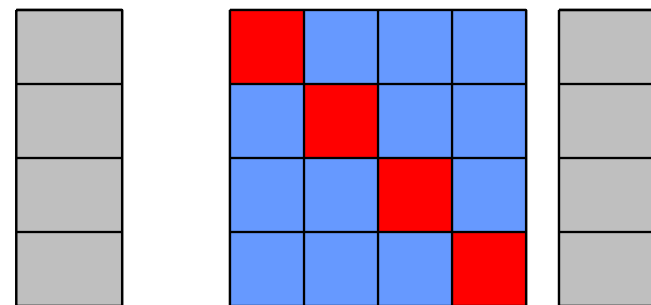
$$\mathbf{Y} = \mathbf{A} \Omega$$


The diagram illustrates the matrix equation $\mathbf{Y} = \mathbf{A} \Omega$. Matrix \mathbf{Y} is a vertical column of 4 gray boxes. Matrix \mathbf{A} is a 4x4 grid with a diagonal of red boxes and off-diagonal blue boxes. Matrix Ω is a vertical column of 4 gray boxes.

Let us consider the problem of finding a basis matrix \mathbf{U}_4 for the block $\mathbf{A}(I_4, I_4^c)$.

Compression of a rank-structured matrix – fully black box

Sample \mathbf{A} with a completely dense random matrix $\Omega \in \mathbb{R}^{N \times (r+m)}$, where m is the leaf node size. (Think $m \approx 2k$ and $r = k + 10$.)

$$\mathbf{Y} = \mathbf{A} \Omega$$


Let us consider the problem of finding a basis matrix \mathbf{U}_4 for the block $\mathbf{A}(I_4, I_4^c)$.

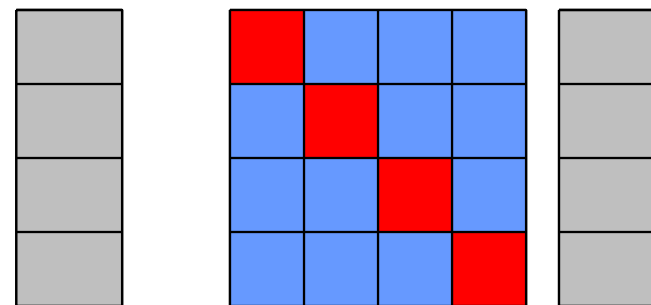
Since $\Omega(I_4, :)$ is of size $m \times (r+m)$, it has a nullspace of dimension at least r . Let

$$\mathbf{Q}_4 = \text{nullspace}(\Omega(I_4, :), r)$$

be an $(r+m) \times r$ orthonormal basis of the nullspace of $\Omega(I_4, :)$.

Compression of a rank-structured matrix – fully black box

Sample \mathbf{A} with a completely dense random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times (r+m)}$, where m is the leaf node size. (Think $m \approx 2k$ and $r = k + 10$.)

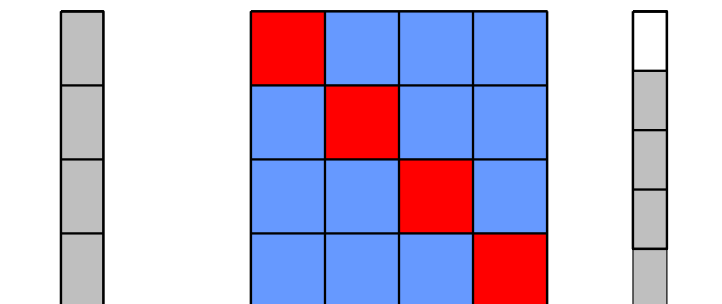
$$\mathbf{Y} = \mathbf{A} \mathbf{\Omega}$$


Let us consider the problem of finding a basis matrix \mathbf{U}_4 for the block $\mathbf{A}(I_4, I_4^c)$.

Since $\mathbf{\Omega}(I_4, :)$ is of size $m \times (r+m)$, it has a nullspace of dimension at least r . Let

$$\mathbf{Q}_4 = \text{nullspace}(\mathbf{\Omega}(I_4, :), r)$$

be an $(r+m) \times r$ orthonormal basis of the nullspace of $\mathbf{\Omega}(I_4, :)$. Then

$$\mathbf{YQ}_4 = \mathbf{A} \mathbf{\OmegaQ}_4$$


Orthonormalizing the sample gives basis matrix \mathbf{U}_4 ,

$$\mathbf{U}_4 = \text{qr}(\mathbf{Y}(I_4, :)\mathbf{Q}_4).$$

Compression of a rank-structured matrix – fully black box

- For each leaf node τ , we compute

$$\mathbf{Q}_\tau = \text{nullspace}(\mathbf{\Omega}(I_\tau, :), r)$$

$$\mathbf{U}_\tau = \text{qr}(\mathbf{Y}(I_\tau, :)\mathbf{Q}_\tau).$$

- \mathbf{U}_τ only depends on $\mathbf{\Omega}(I_\tau, :)$ and $\mathbf{Y}(I_\tau, :)$.
- We only need $r + m$ samples to find \mathbf{U}_τ for every leaf node τ .
- $\mathbf{\Omega}\mathbf{Q}_\tau$ is a Gaussian random matrix, except for the block intentionally zeroed out.

Compression of a rank-structured matrix – fully black box

Recall the telescoping factorization $\mathbf{A} = \mathbf{U}^{(L)} \tilde{\mathbf{A}}^{(L)} (\mathbf{V}^{(L)})^* + \mathbf{D}^{(L)}$.

Steps:

1. Find $\mathbf{U}^{(L)}, \mathbf{V}^{(L)}$.
2. Find $\mathbf{D}^{(L)}$.
3. Compress $\tilde{\mathbf{A}}^{(L)}$ recursively.

Compute randomized samples of \mathbf{A} and \mathbf{A}^ .*

- 1: Form Gaussian random matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ of size $N \times s$.
- 2: Multiply $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$.

Compress level by level from finest to coarsest.

- 3: **for** level $\ell = L, L - 1, \dots, 0$ **do**
- 4: **for** node τ in level ℓ **do**
- 5: **if** τ is a leaf node **then**
- 6: $\mathbf{\Omega}_\tau = \mathbf{\Omega}(I_\tau, :), \quad \mathbf{\Psi}_\tau = \mathbf{\Psi}(I_\tau, :)$
- 7: $\mathbf{Y}_\tau = \mathbf{Y}(I_\tau, :), \quad \mathbf{Z}_\tau = \mathbf{Z}(I_\tau, :)$
- 8: **else**
- 9: Let α and β denote the children of τ .
- $\mathbf{\Omega}_\tau = \begin{bmatrix} \mathbf{V}_\alpha^* \mathbf{\Omega}_\alpha \\ \mathbf{V}_\beta^* \mathbf{\Omega}_\beta \end{bmatrix}, \quad \mathbf{\Psi}_\tau = \begin{bmatrix} \mathbf{U}_\alpha^* \mathbf{\Psi}_\alpha \\ \mathbf{U}_\beta^* \mathbf{\Psi}_\beta \end{bmatrix}$
- $\mathbf{Y}_\tau = \begin{bmatrix} \mathbf{U}_\alpha^* (\mathbf{Y}_\alpha - \mathbf{D}_\alpha \mathbf{\Omega}_\alpha) \\ \mathbf{U}_\beta^* (\mathbf{Y}_\beta - \mathbf{D}_\beta \mathbf{\Omega}_\beta) \end{bmatrix}, \quad \mathbf{Z}_\tau = \begin{bmatrix} \mathbf{V}_\alpha^* (\mathbf{Z}_\alpha - \mathbf{D}_\alpha^* \mathbf{\Psi}_\alpha) \\ \mathbf{V}_\beta^* (\mathbf{Z}_\beta - \mathbf{D}_\beta^* \mathbf{\Psi}_\beta) \end{bmatrix}$
- 10: **if** level $\ell > 0$ **then**
- 11: $\mathbf{Q}_\tau = \text{nullspace}(\mathbf{\Omega}_\tau, r), \quad \mathbf{P}_\tau = \text{nullspace}(\mathbf{\Psi}_\tau, r)$
- 12: $\mathbf{U}_\tau = \text{qr}(\mathbf{Y}_\tau \mathbf{Q}_\tau, r), \quad \mathbf{V}_\tau = \text{qr}(\mathbf{Z}_\tau \mathbf{P}_\tau, r)$
- 13: $\mathbf{D}_\tau = (\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{Y}_\tau \mathbf{\Omega}_\tau^\dagger + \mathbf{U}_\tau \mathbf{U}_\tau^* ((\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*) \mathbf{Z}_\tau \mathbf{\Psi}_\tau^\dagger)^*$
- 14: **else**
- $\mathbf{D}_\tau = \mathbf{Y}_\tau \mathbf{\Omega}_\tau^\dagger$

Compression of a rank-structured matrix – Finding \mathbf{D}

From the telescoping factorization

$$\mathbf{A} = \mathbf{U}^{(L)} \tilde{\mathbf{A}}^{(L)} (\mathbf{V}^{(L)})^* + \mathbf{D}^{(L)}$$

we define $\tilde{\mathbf{A}}^{(L)}$ and $\mathbf{D}^{(L)}$ as follows.

Compression of a rank-structured matrix – Finding \mathbf{D}

From the telescoping factorization

$$\mathbf{A} = \mathbf{U}^{(L)} \tilde{\mathbf{A}}^{(L)} (\mathbf{V}^{(L)})^* + \mathbf{D}^{(L)}$$

we define $\tilde{\mathbf{A}}^{(L)}$ and $\mathbf{D}^{(L)}$ as follows.

$$\mathbf{A} = \mathbf{U}^{(L)} \overbrace{(\mathbf{U}^{(L)})^* \mathbf{A} \mathbf{V}^{(L)}}^{\tilde{\mathbf{A}}^{(L)}} (\mathbf{V}^{(L)})^* + \overbrace{\mathbf{A} - \mathbf{U}^{(L)} (\mathbf{U}^{(L)})^* \mathbf{A} \mathbf{V}^{(L)} (\mathbf{V}^{(L)})^*}_{\mathbf{D}^{(L)}}$$

Compression of a rank-structured matrix – Finding \mathbf{D}

From the telescoping factorization

$$\mathbf{A} = \mathbf{U}^{(L)} \tilde{\mathbf{A}}^{(L)} (\mathbf{V}^{(L)})^* + \mathbf{D}^{(L)}$$

we define $\tilde{\mathbf{A}}^{(L)}$ and $\mathbf{D}^{(L)}$ as follows.

$$\mathbf{A} = \mathbf{U}^{(L)} \overbrace{(\mathbf{U}^{(L)})^* \mathbf{A} \mathbf{V}^{(L)} (\mathbf{V}^{(L)})^*}^{\tilde{\mathbf{A}}^{(L)}} + \overbrace{\mathbf{A} - \mathbf{U}^{(L)} (\mathbf{U}^{(L)})^* \mathbf{A} \mathbf{V}^{(L)} (\mathbf{V}^{(L)})^*}_{\mathbf{D}^{(L)}}$$

Block \mathbf{D}_τ of $\mathbf{D}^{(L)}$ is given by

$$\begin{aligned} \mathbf{D}_\tau &= \mathbf{A}_{\tau,\tau} - \mathbf{U}_\tau \mathbf{U}_\tau^* \mathbf{A}_{\tau,\tau} \mathbf{V}_\tau \mathbf{V}_\tau^* \\ &= \dots \\ &= (\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*) \mathbf{Y}_\tau \boldsymbol{\Omega}_\tau^\dagger + \mathbf{U}_\tau \mathbf{U}_\tau^* \left((\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*) \mathbf{Z}_\tau \boldsymbol{\Psi}_\tau^\dagger \right)^* \end{aligned}$$

Compression of a rank-structured matrix – Compressing $\tilde{\mathbf{A}}^{(L)}$

To compute randomized samples of $\tilde{\mathbf{A}}^{(L)}$, we multiply the telescoping factorization with Ω to obtain

$$\mathbf{Y} = \mathbf{A}\Omega = (\mathbf{U}^{(L)}\tilde{\mathbf{A}}^{(L)}(\mathbf{V}^{(L)})^* + \mathbf{D}^{(L)})\Omega,$$

and rearrange to obtain

$$\underbrace{(\mathbf{U}^{(L)})^*(\mathbf{Y} - \mathbf{D}^{(L)}\Omega)}_{\text{sample matrix}} = \tilde{\mathbf{A}}^{(L)} \underbrace{(\mathbf{V}^{(L)})^*\Omega}_{\text{test matrix}}.$$

Compression of a rank structured matrix: RSRS

Suppose you have extracted samples

$$\mathbf{Y} = \mathbf{A}\mathbf{\Omega} \quad \text{and} \quad \mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}.$$

We use the set $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$ to extract the information to compress the first block using “block nullification” and “block extraction”.

Then do one step of strong recursive skeletonization to obtain a partial factorization

$$\mathbf{A} = \mathbf{L}\tilde{\mathbf{A}}\mathbf{R},$$

where \mathbf{L} and \mathbf{R} each consists of two block elimination steps, and $\tilde{\mathbf{A}}$ is a matrix where some blocks have been zeroed out, and some have been modified.

We next seek a sample of $\tilde{\mathbf{A}}$. To do this, we form

$$\tilde{\mathbf{Y}} := \mathbf{L}^{-1}\mathbf{Y} = \mathbf{L}^{-1}\mathbf{A}\mathbf{\Omega} = \mathbf{L}^{-1}(\mathbf{L}\tilde{\mathbf{A}}\mathbf{R})\mathbf{\Omega} = \tilde{\mathbf{A}}\tilde{\mathbf{\Omega}},$$

where we defined

$$\tilde{\mathbf{\Omega}} := \mathbf{R}\mathbf{\Omega}.$$

Analogously, form $\{\tilde{\mathbf{Z}}, \tilde{\mathbf{\Psi}}\}$.

Proceed to the next box using the set $\{\tilde{\mathbf{Y}}, \tilde{\mathbf{\Omega}}, \tilde{\mathbf{Z}}, \tilde{\mathbf{\Psi}}\}$ in place of $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$.

Key points

- Randomized algorithms for low rank approximation are highly efficient.
 - Interaction with target matrix only through matrix-matrix multiplication → very high practical speed.
 - Particularly efficient for GPUs, out-of-core computing, distributed memory, etc.
 - Structured random maps (“sparse J-L transform”):
Rank- k approximation in complexity $O(mn \log k)$ or even less (vs. $O(mnk)$ for deterministic).
 - Single pass algorithms have been developed for *streaming environments*.
Not possible with deterministic methods!
 - Extension to the case of *tensors* is active area of research.
- Randomized algorithms based on *sampling* make (some) huge problems tractable.
 - Success stories: kernel ridge regression, computational chemistry, tensor approximation, ...
- Randomized compression of global operators.
 - Black box randomized algorithms for compressing global operators have been established.
 - The combination of “fully black box” and “true linear complexity” was realized only recently.
 - Powerful tools in the construction of fast direct solvers for elliptic PDEs.
 - Path to solving multi-physics (and “multi-discretization”) problems.
 - Going from the discrete to the continuum in randomized compression.

Surveys:

- P.G. Martinsson and J. Tropp, “Randomized Numerical Linear Algebra: Foundations & Algorithms”. *Acta Numerica*, 2020. (Arxiv report 2002.01387)

Long survey summarizing major findings in the field in the past decade.

- P.G. Martinsson, “Randomized methods for matrix computations.” *The Mathematics of Data*, IAS/Park City Mathematics Series, 25(4), pp. 187 - 231, 2018.
Book chapter that is written to be accessible to a broad audience. Focused on practical aspects.
- N. Halko, P.G. Martinsson, J. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.” *SIAM Review*, 53(2), 2011, pp. 217-288.
Survey that describes the randomized SVD and its variations.

Tutorials, summer schools, etc:

- 2020: 3 lecture mini course on randomized linear algebra, KTH, Stockholm. Videos available.
- 2016: Park City Math Institute (IAS): *The Mathematics of Data*.
- 2014: CBMS summer school at Dartmouth College. 10 lectures on YouTube.
- 2009: NIPS tutorial lecture, Vancouver, 2009. Online video available.

Software:

- ID: <http://tygert.com/software.html> (ID, SRFT, CPQR, etc)
- RSVDPACK: <https://github.com/sergeyvoronin> (RSVD, randomized ID and CUR)
- HQRRP: <https://github.com/flame/hqrrp/> (LAPACK compatible randomized CPQR)
- Randomized UTV: <https://github.com/flame/randutv>

DOE report on randomized algorithms: <https://arxiv.org/abs/2104.11079> (2021)