Randomized algorithms for linear algebraic computations

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

- **Randomized dimension reduction – the basics** *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.

- **Recent advances**
  - A posteriori error estimation; rank adaptivity; sketch recycling.
  - Bring it all together: IterativeCUR

- **Sampling based methods for huge problems** *Very brief!*
  - Monte Carlo style methods $\rightarrow$ less reliable, less accurate, less robust.
  - Enable the solution of stupendously large problems that would otherwise be intractable.
  - Resolved some long-standing open theoretical questions in linear algebra.

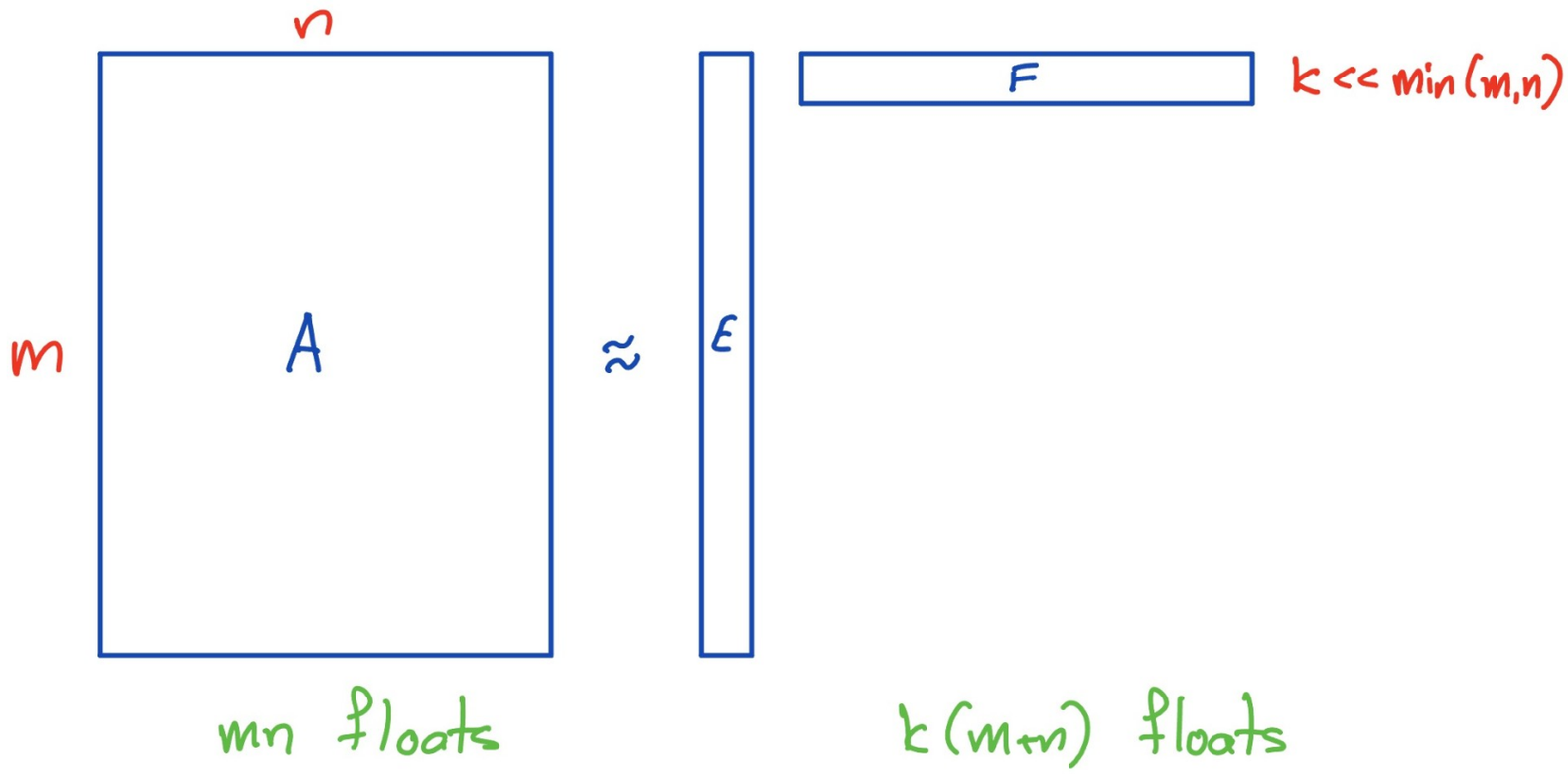
- **Randomized methods for differential and integral operators** *Time permitting . . .*
  - Approximation of global operators of mathematical physics (solution operators, DtNs, . . .).
  - Tools for matrices that are not of global low rank, but have structure that can be exploited.
  - Conceptually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

Low rank approximation:

Let **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 **E** and **F** such that

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}^*}.$$



Low rank approximation:

Let **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 **E** and **F** such that

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{E}} \quad \underset{k \times n}{\mathbf{F}^*}.$$

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:

Excellent algorithms for numerical low rank approximation 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, $\text{ran}(\mathbf{Q}) = \text{ran}\big(\big[\mathbf{b},\, \mathbf{Ab},\, \mathbf{A}^2\mathbf{b},\, \ldots,\, \mathbf{A}^{k-1}\mathbf{b}\big]\big)$.
  - 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.

Randomized algorithms for low rank approximation:

Problem: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, where $k \ll \min(m, n)$, we seek to compute an approximate partial singular value decomposition:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^*, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

with $\mathbf{U}$ and $\mathbf{V}$ having orthonormal columns, and $\mathbf{D}$ diagonal.

Solution:

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{G}$. `G = randn(n,k)`
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{AG}$. `Y = A * G`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\text{col}(\mathbf{Y}) = \text{col}(\mathbf{Q})$. `[Q, ~] = qr(Y)`
4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`
5. Compute the SVD of $\mathbf{B}$ (small!): $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

Why does it work? When $\mathbf{A}$ has exact rank k, the algorithm succeeds with probability 1.

Randomized algorithms for low rank approximation:

Problem: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, where $k \ll \min(m, n)$, we seek to compute an approximate partial singular value decomposition:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^*, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

with $\mathbf{U}$ and $\mathbf{V}$ having orthonormal columns, and $\mathbf{D}$ diagonal.

Solution:

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{G}$. `G = randn(n,k)`
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{AG}$. `Y = A * G`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\text{col}(\mathbf{Y}) = \text{col}(\mathbf{Q})$. `[Q, ~] = qr(Y)`
4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`
5. Compute the SVD of $\mathbf{B}$ (small!): $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

Why does it work? When $\mathbf{A}$ has exact rank k, the algorithm succeeds with probability 1. In the general case, it works very well when the singular values decay rapidly. If they decay slowly, accuracy deteriorates.

Randomized algorithms for low rank approximation:

Problem: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, where $k \ll \min(m, n)$, we seek to compute an approximate partial singular value decomposition:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^*, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

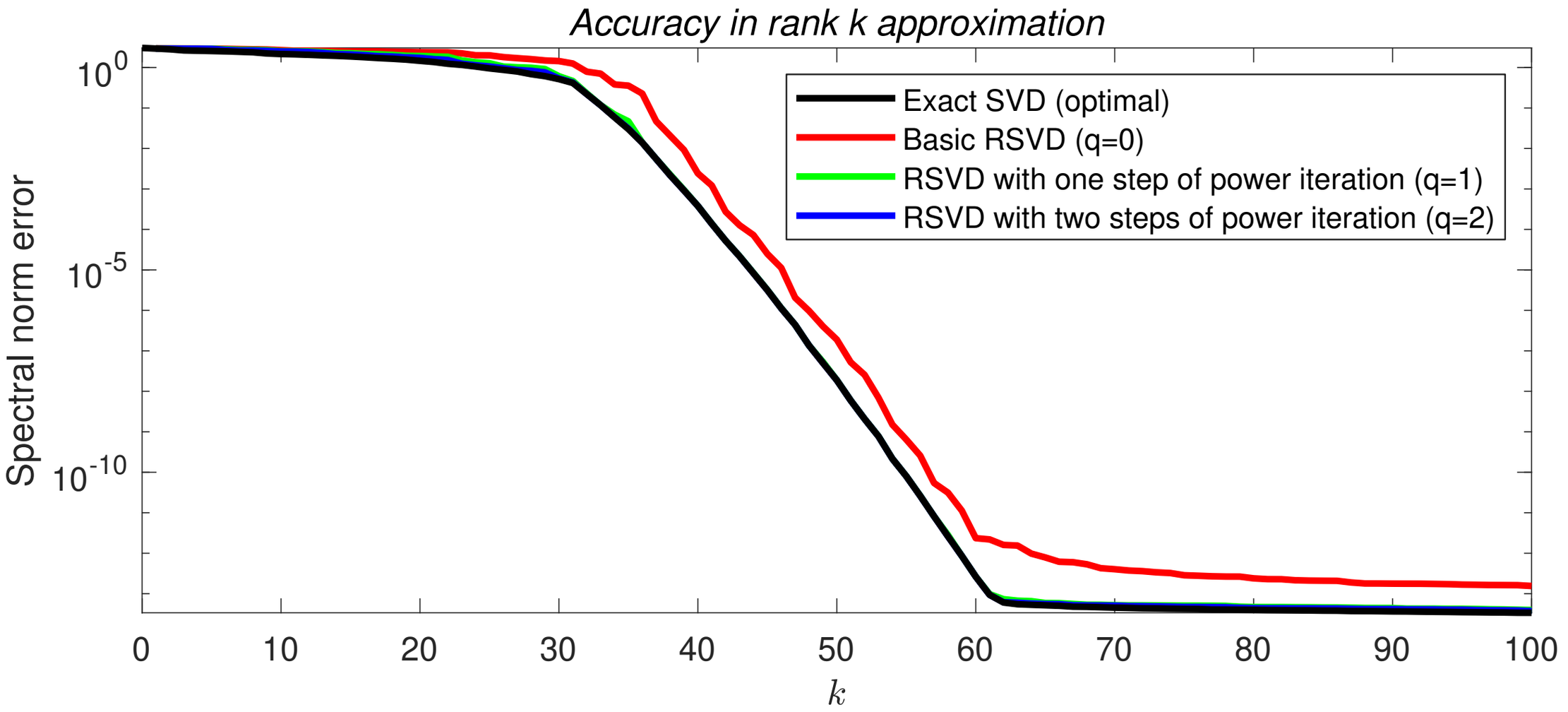
with $\mathbf{U}$ and $\mathbf{V}$ having orthonormal columns, and $\mathbf{D}$ diagonal.

Solution:

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{G}$. `G = randn(n,k)`
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{AG}$. `Y = A * G`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\text{col}(\mathbf{Y}) = \text{col}(\mathbf{Q})$. `[Q, ~] = qr(Y)`
4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`
5. Compute the SVD of $\mathbf{B}$ (small!): $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

Power iteration: When the singular values of $\mathbf{A}$ decay slowly, precision can be improved by replacing the formula $\mathbf{Y} = \mathbf{AG}$ on line 2 by $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*\mathbf{G})$, or $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*(\mathbf{AG}))$, or ...

Randomized low rank approximation:



The plot shows the errors from the randomized range finder. To be precise, we plot

$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

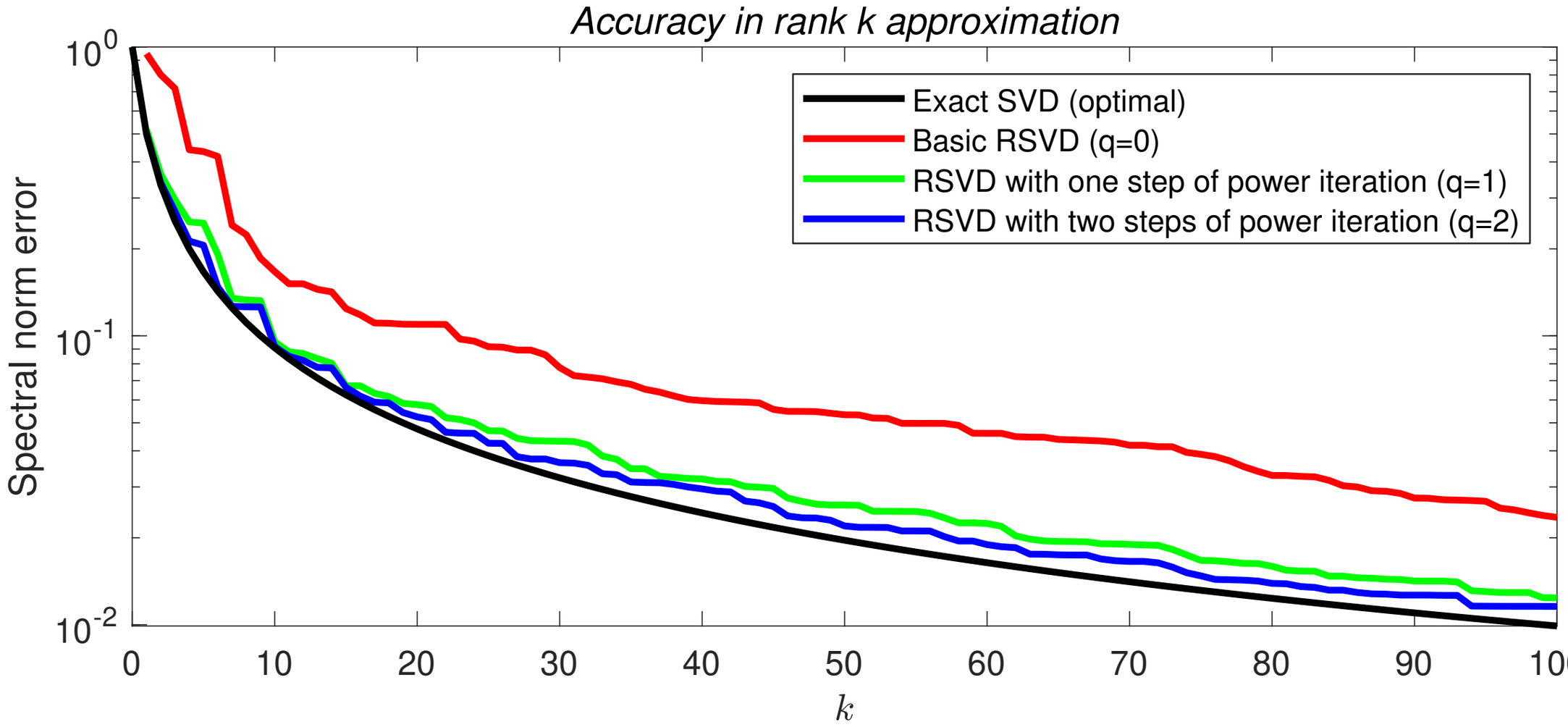
where $\mathbf{P}_k$ is the orthogonal projection onto the first k columns of

$$\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q\mathbf{A}\mathbf{G},$$

and where $\mathbf{G}$ is a Gaussian random matrix.

The matrix $\mathbf{A}$ is an approximation to a scattering operator for a Helmholtz problem.

Randomized low rank approximation:



The plot shows the errors from the randomized range finder. To be precise, we plot

$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

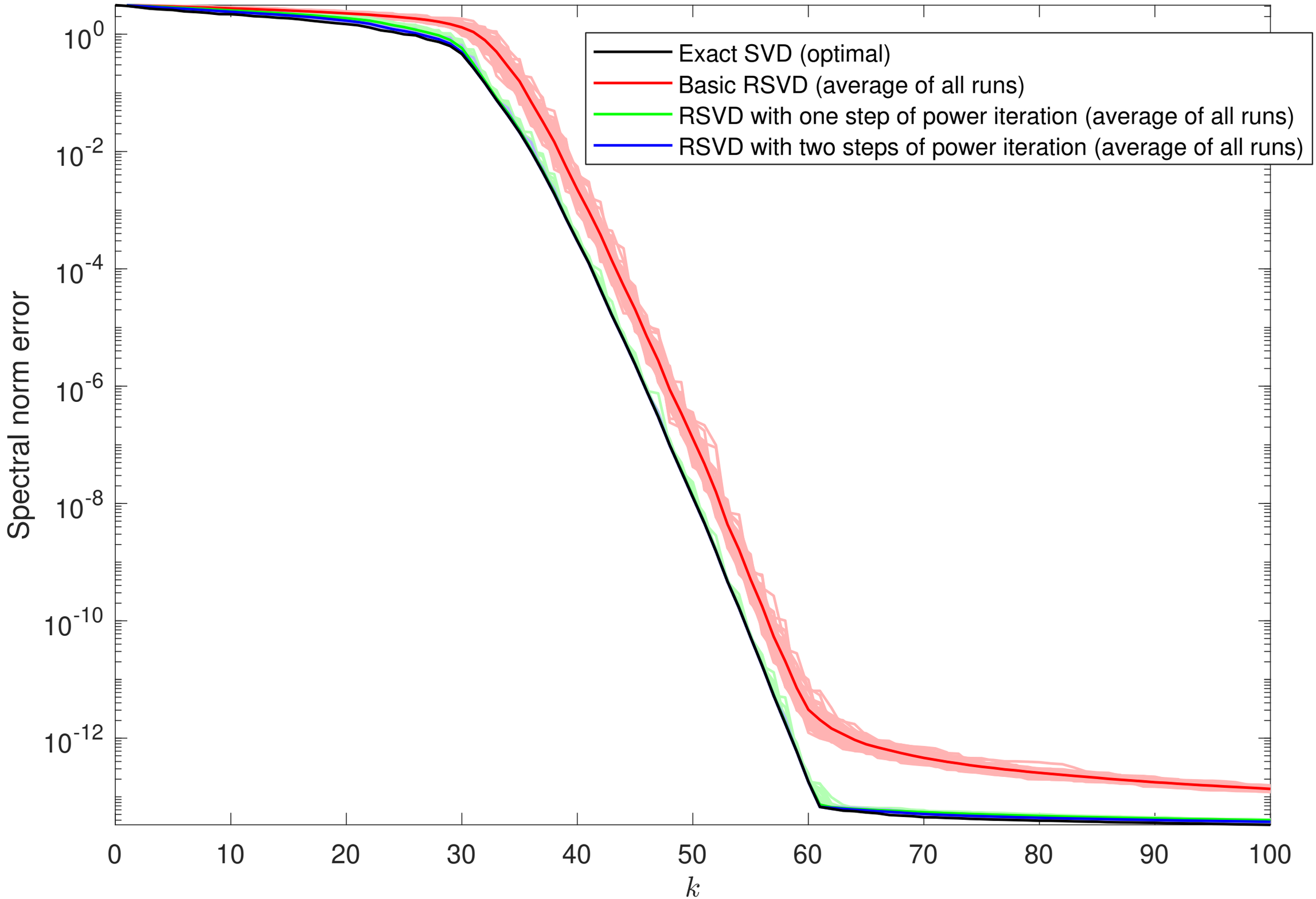
where $\mathbf{P}_k$ is the orthogonal projection onto the first k columns of

$$\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q\mathbf{A}\mathbf{G},$$

and where $\mathbf{G}$ is a Gaussian random matrix.

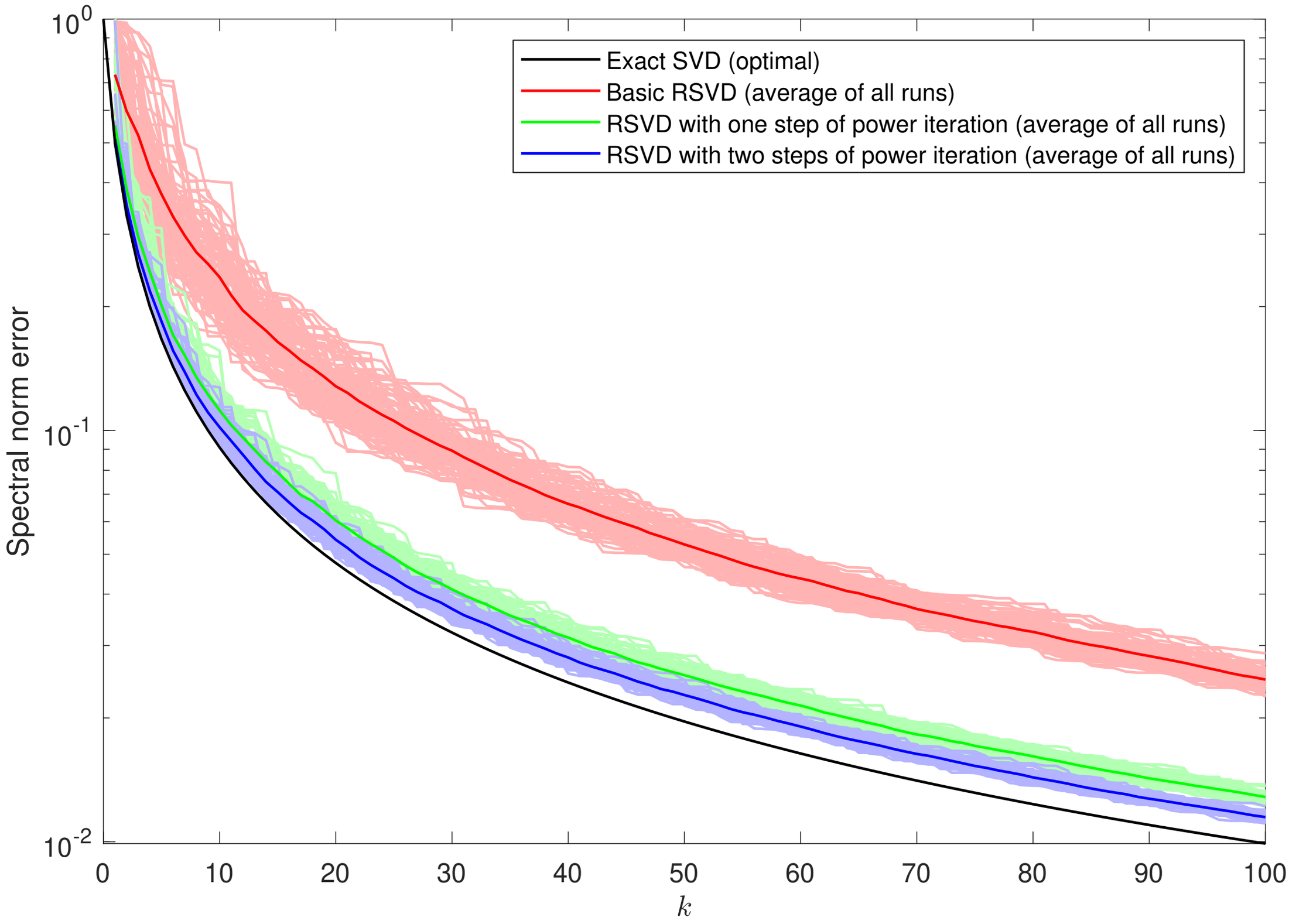
The matrix $\mathbf{A}$ now has singular values that decay slowly.

Randomized low rank approximation: The same plot, but showing 100 instantiations.



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

Randomized low rank approximation: The same plot, but showing 100 instantiations.



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

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 $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Error analysis: Observe first that steps (3)–(6) are all exact (up to floating point arithmetic). The error is entirely determined by the draw of the matrix $\mathbf{Y}$, since

$$\mathbf{A}_{\text{approx}} = \mathbf{UDV}^* = \mathbf{Q}\hat{\mathbf{U}}\mathbf{DV}^* = \mathbf{QQ}^*\mathbf{A} = \mathbf{YY}^\dagger\mathbf{A}.$$

So:

$$\|\mathbf{A} - \mathbf{A}_{\text{approx}}\| = \|\mathbf{A} - \mathbf{YY}^\dagger\mathbf{A}\|.$$

In other words, the question is how well the columns of $\mathbf{Y}$ span the column space of $\mathbf{A}$.

We recall from the Eckart-Young theorem that the *optimal* basis for col($\mathbf{A}$) is given by the dominant k left singular vectors of $\mathbf{A}$. Let $\mathbf{A} = \tilde{\mathbf{U}}\tilde{\mathbf{D}}\tilde{\mathbf{V}}^*$ be the exact SVD of $\mathbf{A}$. Then

$$\mathbf{Y} = \mathbf{AG} = \tilde{\mathbf{U}}\tilde{\mathbf{D}}\tilde{\mathbf{V}}^*\mathbf{G} = \{\text{Set } \tilde{\mathbf{G}} := \tilde{\mathbf{V}}^*\mathbf{G}\} = \tilde{\mathbf{U}}\tilde{\mathbf{D}}\tilde{\mathbf{G}},$$

where $\tilde{\mathbf{G}}$ also has a Gaussian distribution. So the j'th column of $\mathbf{Y}$ is

$$\mathbf{y}(:,j) = \sum_{i=1}^{n} \tilde{\sigma}_i \, \tilde{g}_{i,j} \, \tilde{\mathbf{u}}_i.$$

Input: An $m \times n$ matrix **A**, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Recall that the j'th column of **Y** satisfies

$$\mathbf{y}(:,j) = \sum_{i=1}^{n} \tilde{\sigma}_i \, \tilde{g}_{i,j} \, \tilde{\mathbf{u}}_i.$$

where $\tilde{\sigma}_j$ is the j'th exact singular value, where $\tilde{\mathbf{u}}_j$ is the associated left singular vector, and where $\tilde{g}_{i,j}$ are i.i.d. $\in \mathcal{N}(0,1)$.

- When **A** has *exact* rank k, the algorithm succeeds (i.e. $\mathbf{A} = \mathbf{QQ}^*\mathbf{A}$) 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 explains why powering helps in cases where the singular values decay slowly, since this boosts the separation between large/small svds. For instance, replacing $\mathbf{Y} = \mathbf{A\Omega}$ by $\mathbf{Y} = \mathbf{AA}^*\mathbf{A\Omega}$ results in $\mathbf{y}_j = \sum_{i=1}^{n} \left(\sigma_i^3 \psi_{i,j}\right) \mathbf{u}_i$.

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 $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

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.

Input: An $m \times n$ matrix **A**, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

To illustrate, suppose that we seek to match the optimal error in a rank-k approximation.

We apply the basic randomized procedure involving $\ell := k + p$ samples.

1. Draw an $n \times \ell$ Gaussian random matrix $\mathbf{\Omega}$. `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 **Q** such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

The resulting error then satisfies

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

where "$\mathbf{\Psi}_{i,j}$" denotes an $i \times j$ matrix drawn from a Gaussian distribution, and where $\mathbf{D}_2 = \text{diag}(\sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_{\min(m,n)})$ holds the "tail" singular values of **A**.

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

The suboptimality term depends on $\sigma_{\min}(\mathbf{\Psi}_{k,\ell})$, where $\mathbf{\Psi}_{k,\ell}$ 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?

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 $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

To illustrate, suppose that we seek to match the optimal error in a rank-k approximation.

We apply the basic randomized procedure involving $\ell := k + p$ samples.

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

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

where "$\mathbf{\Psi}_{i,j}$" denotes an $i \times j$ matrix drawn from a Gaussian distribution, and where $\mathbf{D}_2 = \text{diag}(\sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_{\min(m,n)})$ 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 $\left(\mathbb{E}\left[\|\mathbf{\Psi}^\dagger\|_{\text{F}}^2\right]\right)^{1/2} \leq \sqrt{\frac{k}{p-1}}$ and $\mathbb{E}\left[\|\mathbf{\Psi}^\dagger\|\right] \leq \frac{e\sqrt{k+p}}{p}$.

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 $\mathbf{G}$.

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

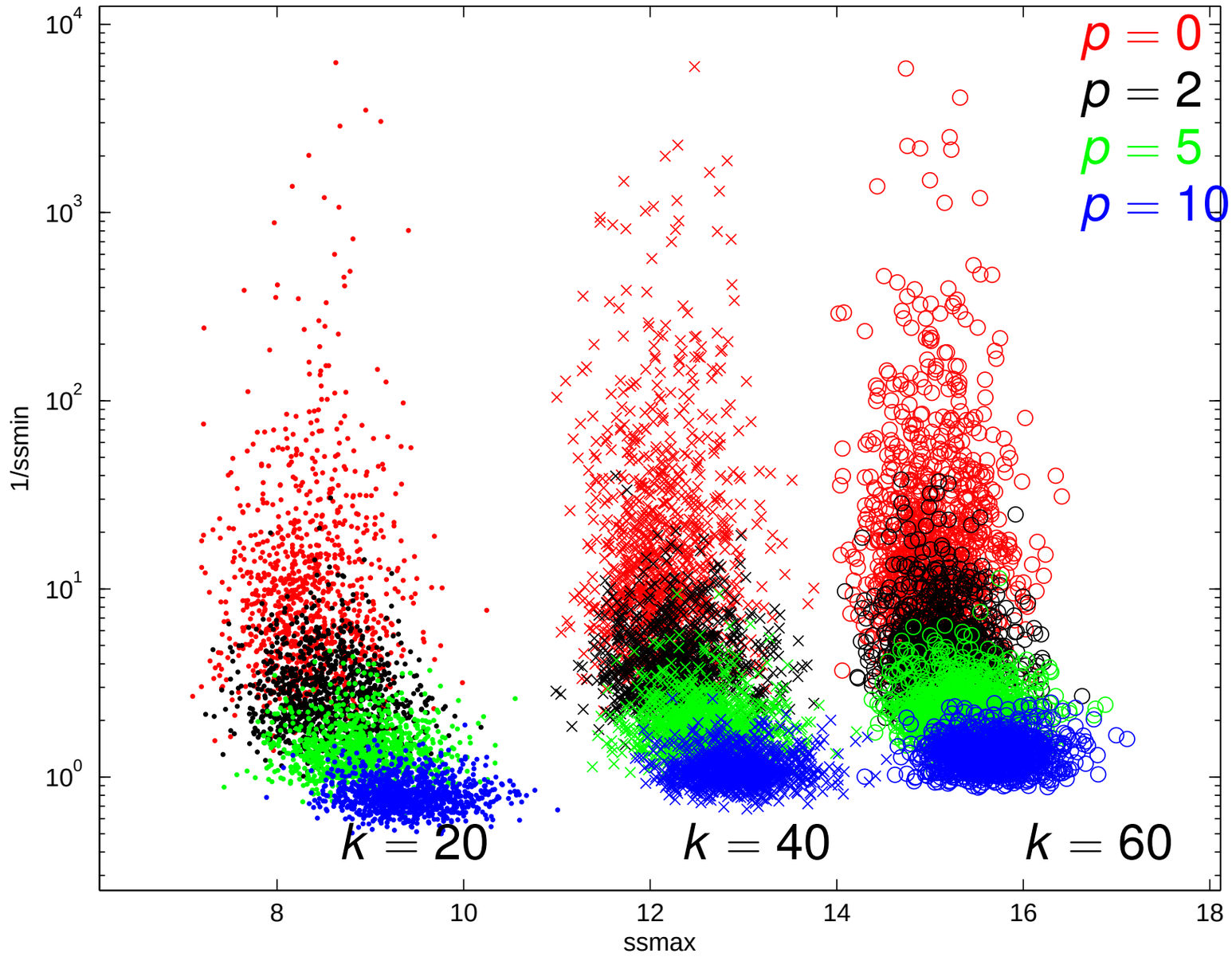
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.

(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 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

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



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

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 $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Using the bound $\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\mathbf{\Psi}_{(n-k),\ell}\mathbf{\Psi}_{k,\ell}^{\dagger}\|^2$, along with the bounds on $\sigma_{\min}(\Psi_{k,\ell})$, 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 $\mathbf{\Omega}$ be a Gaussian random matrix of size $n \times (k+p)$ and set $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$. Then the average error satisfies

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

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

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!)

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 $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key points:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.
- Communication efficient: Out-of-core, GPU, distributed memory, ...

Input: An $m \times n$ matrix **A**, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key points:

- High practical speed — interacts with **A** only through matrix-matrix multiplication.
- Communication efficient: Out-of-core, GPU, distributed memory, ...
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn)$.

The key is to use a *Fast Johnson-Lindenstrauss transform*.

- 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 Ω. (Say $r = 2$ or $r = 4$.) Cost is now $O(mn)$!

Practical acceleration is achieved at ordinary matrix sizes.

(Some additional modifications are required – we sketch from both sides, etc.)

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

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

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

(2) Form th

(3) Compu

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

$\mathbf{\hat{U}DV}^*$.

Key points:

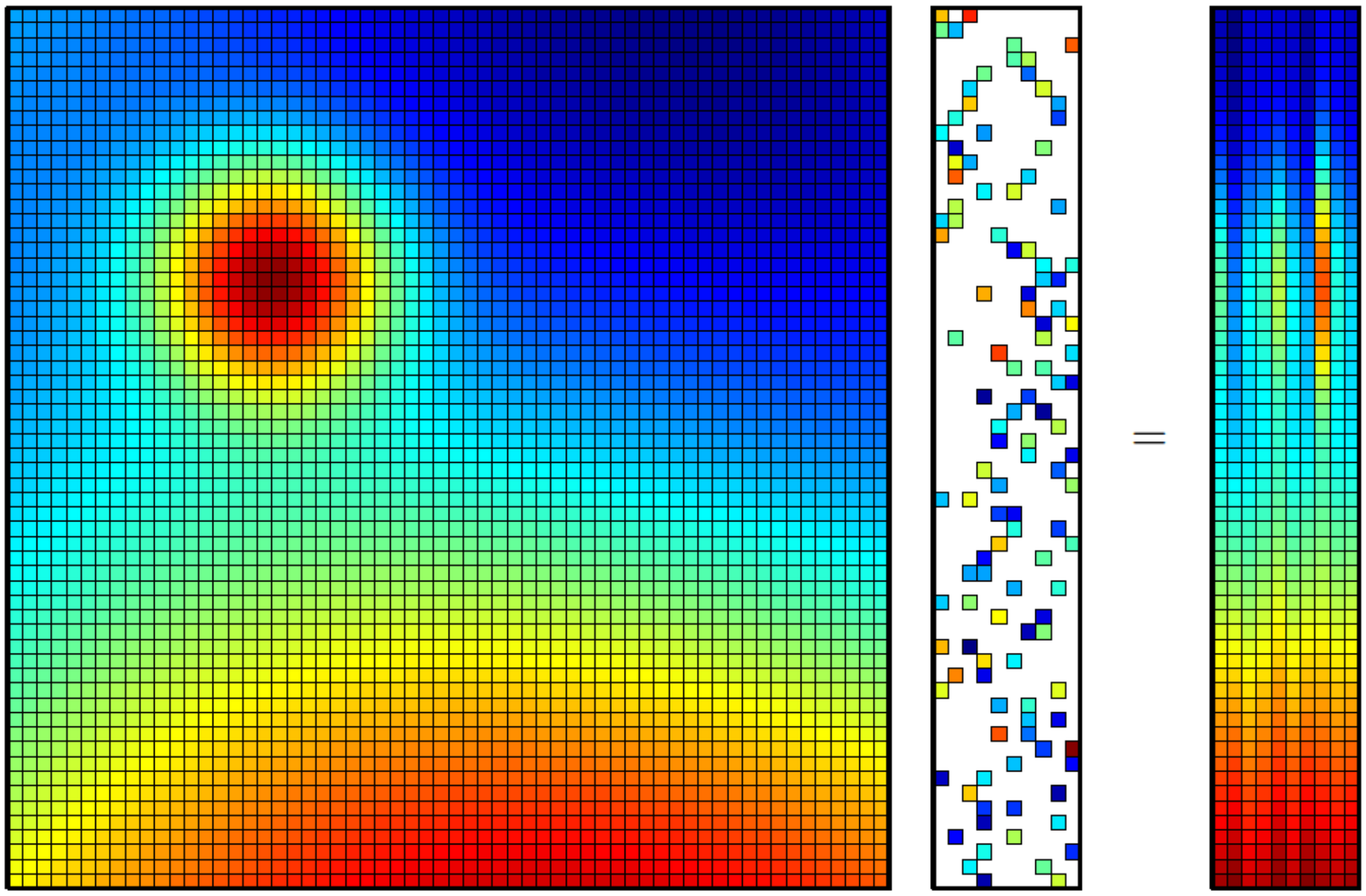
- High pra on.
- Commu
- Conside of a dense n

The key

- Rando
- Chains
- "Spars

  Cost is

Practica



(Some additional modifications are required – we sketch from both sides, etc.)

Input: An $m \times n$ matrix **A**, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key points:

- High practical speed — interacts with **A** only through matrix-matrix multiplication.
- Communication efficient: Out-of-core, GPU, distributed memory, ...
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn)$.

  The key is to use a *Fast Johnson-Lindenstrauss transform*.
  - 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 Ω. (Say $r = 2$ or $r = 4$.)
    Cost is now $O(mn)$!
- Single pass algorithms have been developed for *streaming environments.*

  The idea is that you are allowed to observe each matrix element only once.

  You cannot store the matrix. *Not possible with deterministic methods!*

Random mixing for solving linear systems & least squares problems

Fast random transforms open the door to 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 $\mathbf{\Omega}$ 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*

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

Again, the point is communication efficiency. *(Parker 1995)*

Random mixing for solving linear systems & least squares problems

Fast random transforms open the door to 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 $\mathbf{\Omega}$ 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)*

General idea:

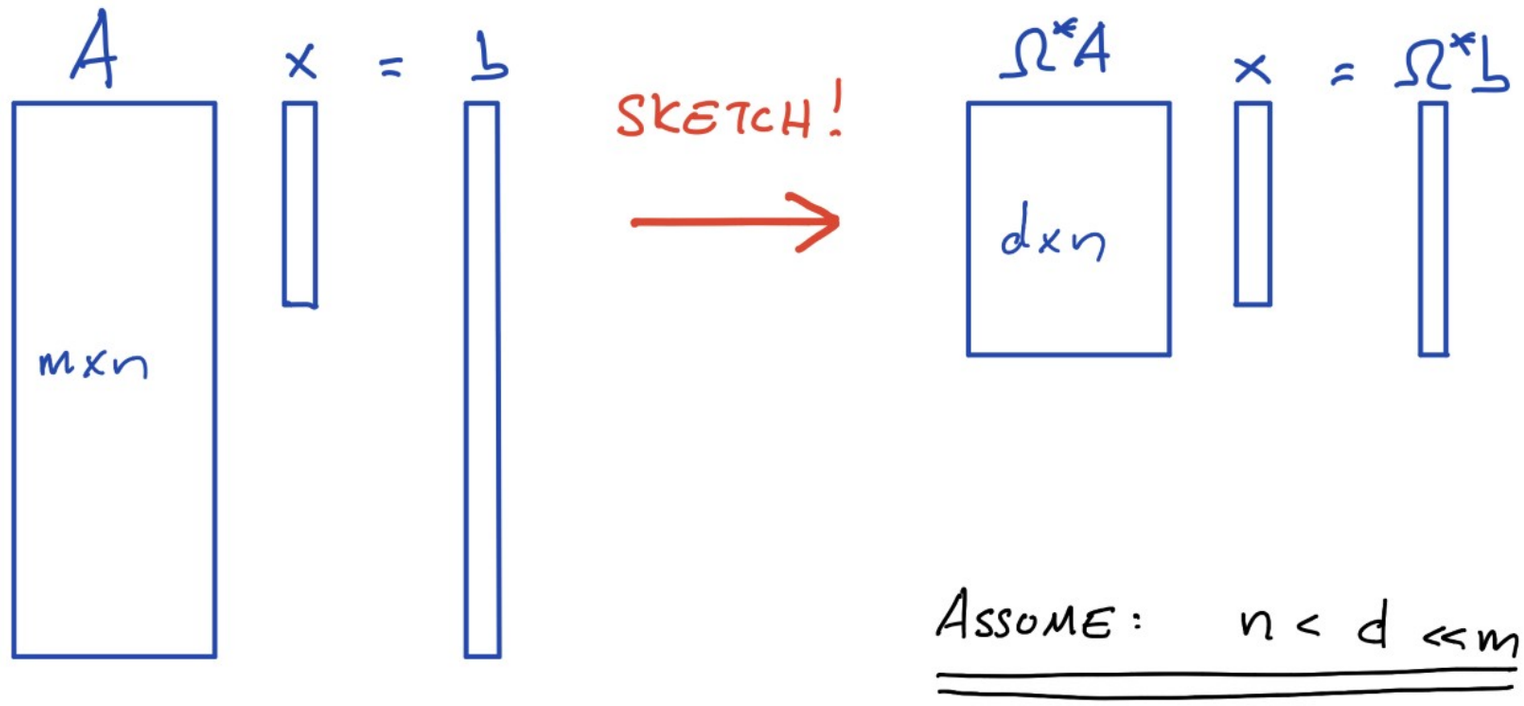
Randomized mixing puts you in a "generic position".

Standard Gram-Schmidt *without pivoting* becomes a reliable way to build an orthonormal basis.

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 $\mathbf{\Omega} \in \mathbb{R}^{m \times d}$ 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 $\mathbf{\Omega}^*\mathbf{A}$ so that $\mathbf{\Omega}^*\mathbf{A} = \mathbf{QM}$.

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

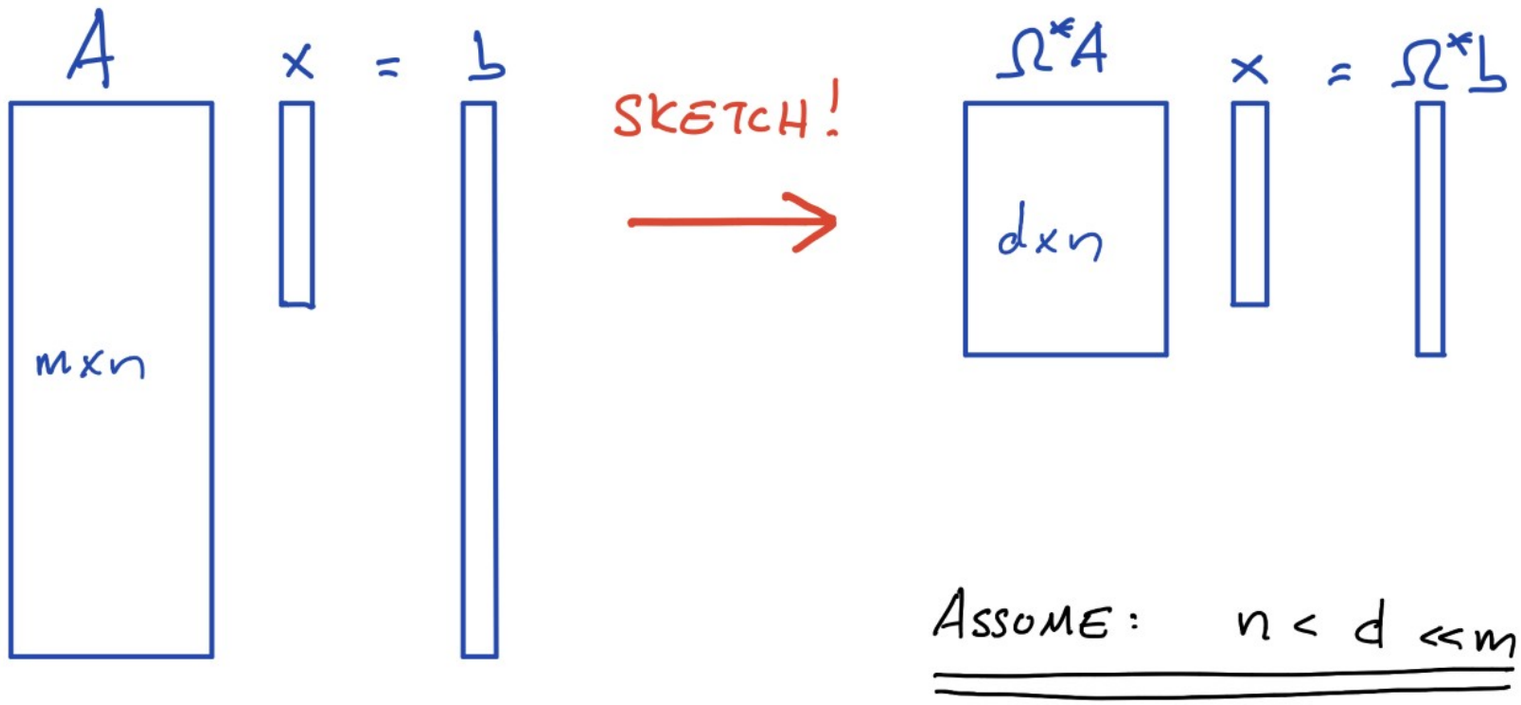
Ideal use of randomization. Guaranteed accuracy if you evaluate the residual.

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 $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$.

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

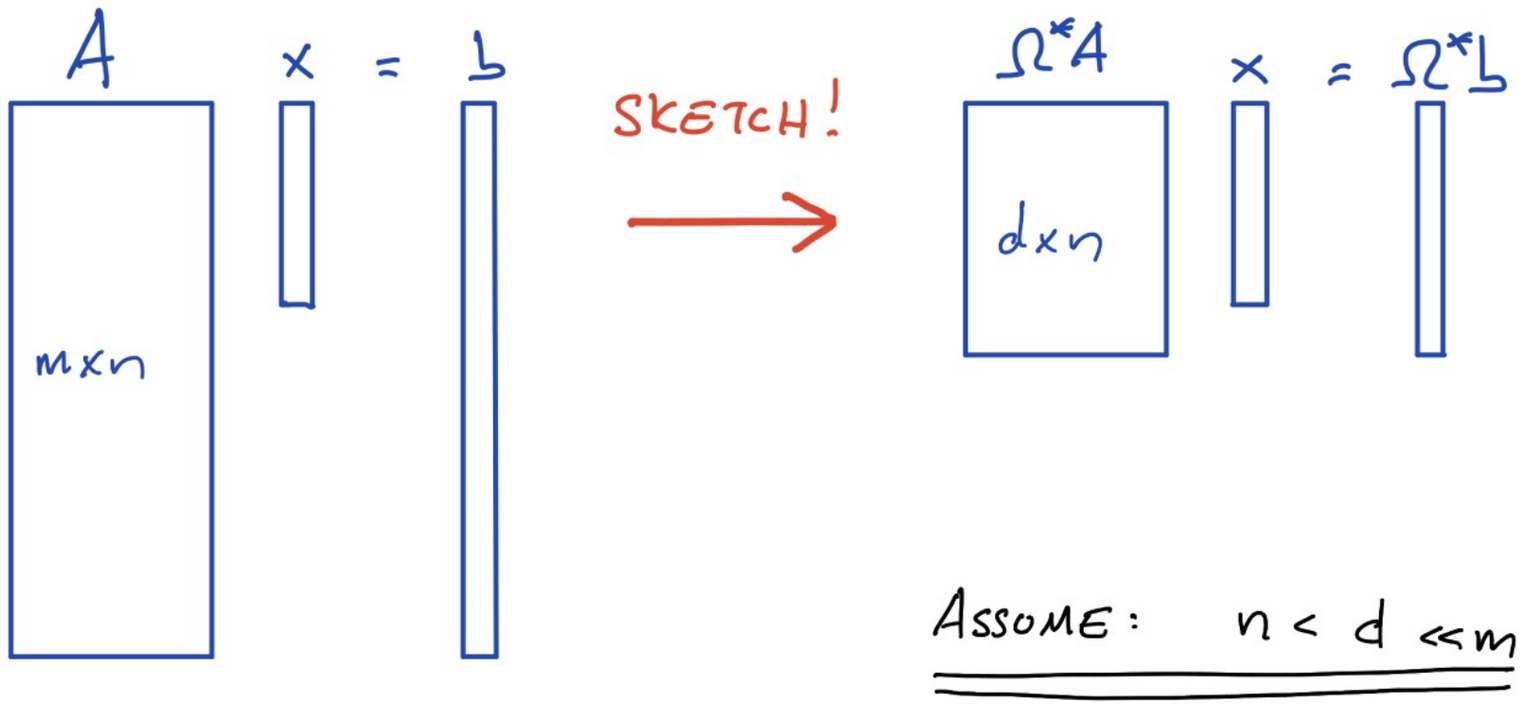


Question: Can the sketch be *down-sampling*? Simply pick d equations randomly?

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 $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$.

Draw a random embedding $\mathbf{\Omega} \in \mathbb{R}^{m \times d}$ 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

- **Randomized dimension reduction – the basics** *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.

- **Recent advances**
  - A posteriori error estimation; rank adaptivity; sketch recycling.
  - Bring it all together: IterativeCUR

- **Sampling based methods for huge problems**
  - Monte Carlo style methods $\rightarrow$ less reliable, less accurate, less robust.
  - Enable the solution of stupendously large problems that would otherwise be intractable.
  - Resolved some long-standing open theoretical questions in linear algebra.

- **Randomized methods for differential and integral 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 ...

Recent work 1 (out of 5): A posteriori error estimation

Question: How estimate the error in a specific instantiation of a randomized algorithm?

Our framing is that we are given an $m \times n$ matrix $\mathbf{A}$, and use the randomized range finder to build an approximation:

- Draw an $n \times \ell$ random matrix $\mathbf{\Omega}$, for say $\ell = k + 10$ or $\ell = 2k$.
  Note: We may use a very fast, but possibly less reliable, class of random matrices.
- Form $\mathbf{Y} = \mathbf{A\Omega}$, and perform QR factorization $[\mathbf{Q}, \sim] = \mathtt{qr}(\mathbf{Y}, 0)$.
- Use $\mathbf{A}_{\text{approx}} = \mathbf{Q}\left(\mathbf{Q}^*\mathbf{A}\right)$ as the approximation.

Our objective is to estimate the error

$$e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\|$$

using only the information at hand!

Why?

- Enable the use of "risky" random dimension reducing maps.
  Cf. the "responsibly reckless" mode of computing, in Dongarra's terminology.
- The numerical rank of $\mathbf{A}$ may not be known in advance.
- Need bounds and estimates that involve *only quantities actually at hand.*

Recent work 1 (out of 5): A posteriori error estimation

Question: How estimate the error in a specific instantiation of a randomized algorithm?

Answer: You can very inexpensively compute a *certificate of accuracy*. To illustrate, consider the RSVD: We draw a random matrix $\mathbf{\Omega}$, set $\mathbf{Y} = \mathbf{A\Omega}$, and approximate $\mathbf{A}$ via

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

where $[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{A}, 0)$. We seek to estimate the error

$$e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}.$$

The idea is to draw a thin slice of a Gaussian, $\mathbf{G} \in \mathbb{R}^{n\times q}$, for $q = 10$ say, that is *quarantined* from the construction of $\mathbf{A}_{\text{approx}}$ and is used purely to estimate the error. Evaluate $\mathbf{C} = \mathbf{AG}$. Then use that for any matrix $\mathbf{H}$, we have

$$\mathbb{E}\big[\|\mathbf{HG}\|_{\text{F}}^2\big] = q\|\mathbf{H}\|_{\text{F}}^2.$$

Setting $\mathbf{H} = \mathbf{A} - \mathbf{A}_{\text{approx}}$, we use the estimate

$$\|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}^2 \approx \frac{1}{q}\|(\mathbf{A} - \mathbf{A}_{\text{approx}})\mathbf{G}\|_{\text{F}}^2 = \frac{1}{q}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{AG}\|_{\text{F}}^2 = \frac{1}{q}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{C}\|_{\text{F}}^2.$$

Very reliable. Inexpensive. *Martinsson, Tropp, Acta Numerica 2020, Sec. 12.2.*

We can do better still. Observe that e does not depend on $\mathbf{Q}$, since $\mathbf{A}_{\text{approx}} = \mathbf{Y}\left(\mathbf{Y}^{\dagger}\mathbf{A}\right)$.

Recent work 1 (out of 5): A posteriori error estimation

Question: How estimate the error in a specific instantiation of a randomized algorithm?

Recall from previous slide: Draw a random matrix $\mathbf{\Omega}$, set $\mathbf{Y} = \mathbf{A\Omega}$, approximate $\mathbf{A}$ via $\mathbf{A}_{\text{approx}} = \mathbf{QQ}^*\mathbf{A}$, where $[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y}, 0)$. We seek to estimate $e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}$. To get a *certificate of accuracy*, we draw $\mathbf{G}$, set $\mathbf{C} = \mathbf{AG}$, and use $e^2 \approx \frac{1}{q}\|(\mathbf{I} - \mathbf{QQ}^*)\mathbf{C}\|_{\text{F}}^2$.

Recent work 1 (out of 5): A posteriori error estimation

Question: How estimate the error in a specific instantiation of a randomized algorithm?

Recall from previous slide: Draw a random matrix $\boldsymbol{\Omega}$, set $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$, approximate $\mathbf{A}$ via $\mathbf{A}_{\text{approx}} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$, where $[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y}, 0)$. We seek to estimate $e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}$. To get a *certificate of accuracy*, we draw $\mathbf{G}$, set $\mathbf{C} = \mathbf{A}\mathbf{G}$, and use $e^2 \approx \frac{1}{q}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{C}\|_{\text{F}}^2$.

Acceleration via fast sketching: Observe that $\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{C}\|_{\text{F}}^2$ is the minimal error when seeking to fit $\mathbf{C}$ in $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{A}\boldsymbol{\Omega}) = \text{ran}(\mathbf{Y})$. So

$$\|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}^2 \approx \frac{1}{q}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{C}\|_{\text{F}}^2 = \frac{1}{q} \inf_{\mathbf{M} \in \mathbb{R}^{\ell \times q}} \|\mathbf{Y}\mathbf{M} - \mathbf{C}\|_{\text{F}}^2.$$

Next, we *sketch* the least squares problem. Draw FJLT $\boldsymbol{\Psi} \in \mathbb{R}^{m \times s}$, with $s = O(k)$, then

$$\|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}^2 \approx \frac{1}{q}\frac{m}{s} \inf_{\mathbf{M} \in \mathbb{R}^{\ell \times q}} \|\boldsymbol{\Psi}^*\big(\mathbf{Y}\mathbf{M} - \mathbf{C}\big)\|_{\text{F}}^2.$$

So in the end, all we need to do is to approximately solve the least squares problem

$$\underset{s \times \ell}{\big(\boldsymbol{\Psi}^*\mathbf{Y}\big)} \quad \underset{\ell \times q}{\mathbf{M}} \quad = \underset{s \times q}{\big(\boldsymbol{\Psi}^*\mathbf{C}\big)}$$

The total extra cost (beyond the cost of RSVD):

- q extra matvecs. (Think $q = 10$.) Can be done as a block.
- Sketching step – evaluate $\boldsymbol{\Psi}^*\mathbf{Y}$ and $\boldsymbol{\Psi}^*\mathbf{C}$. Cost $O(m\ell)$.
- Solve small least squares problem. Cost $O(s\,\ell^2) = O(k^3)$.

Recent work 1 (out of 5): A posteriori error estimation

Question: How estimate the error in a specific instantiation of a randomized algorithm?

Recall from previous slide: Draw a random matrix $\boldsymbol{\Omega}$, set $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$, approximate $\mathbf{A}$ via $\mathbf{A}_{\text{approx}} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$, where $[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y}, 0)$. We seek to estimate $e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}$. To get a *certificate of accuracy*, we draw $\mathbf{G}$, set $\mathbf{C} = \mathbf{A}\mathbf{G}$, and use $e^2 \approx \frac{1}{q}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{C}\|_{\text{F}}^2$.

Acceleration via fast sketching: Observe that $\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{C}\|_{\text{F}}^2$ is the minimal error when seeking to fit $\mathbf{C}$ in $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{A}\boldsymbol{\Omega}) = \text{ran}(\mathbf{Y})$. So

$$\|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}^2 \approx \frac{1}{q}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{C}\|_{\text{F}}^2 = \frac{1}{q} \inf_{\mathbf{M} \in \mathbb{R}^{\ell \times q}} \|\mathbf{Y}\mathbf{M} - \mathbf{C}\|_{\text{F}}^2.$$

Next, we *sketch* the least squares problem. Draw FJLT $\boldsymbol{\Psi} \in \mathbb{R}^{m \times s}$, with $s = O(k)$, then

$$\|\mathbf{A} - \mathbf{A}_{\text{approx}}\|_{\text{F}}^2 \approx \frac{1}{q}\frac{m}{s} \inf_{\mathbf{M} \in \mathbb{R}^{\ell \times q}} \|\boldsymbol{\Psi}^*\big(\mathbf{Y}\mathbf{M} - \mathbf{C}\big)\|_{\text{F}}^2.$$

So in the end, all we need to do is to approximately solve the least squares problem

$$\underset{s \times \ell}{\big(\boldsymbol{\Psi}^*\mathbf{Y}\big)} \quad \underset{\ell \times q}{\mathbf{M}} \quad = \underset{s \times q}{\big(\boldsymbol{\Psi}^*\mathbf{C}\big)}$$

- *Very* cheap error estimation. High accuracy.
- Involves only available data.
- No need to even form $\mathbf{Q}$! Can be done as soon as $\mathbf{A}\boldsymbol{\Omega}$ is available.

With Yuji Nakatsukasa. Forthcoming.

Recent work 1 (out of 5): A posteriori error estimation

Additional compelling techniques presented in:

E.N. Epperly, J. Tropp, “Efficient Error and Variance Estimation for Randomized Matrix Computations”, SISC. **46**(1), 2024.

“Leave-one-out” error estimator for randomized low-rank approximations.

“Jackknife resampling method” for estimating the variance of the output of a number of randomized matrix computations.

Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

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, $k = 2, 4, 8, 16, \ldots$

Gives correct asymptotics. But quite wasteful and slow.

Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

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

Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

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 **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.

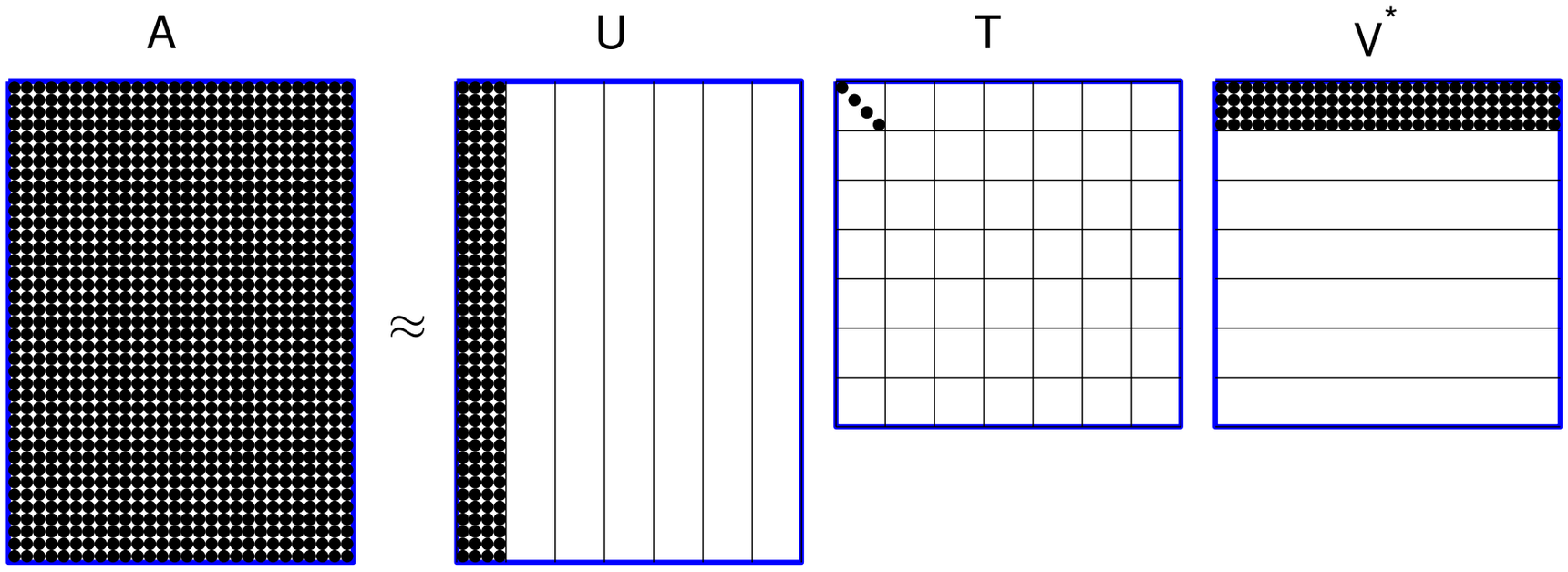
Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

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.

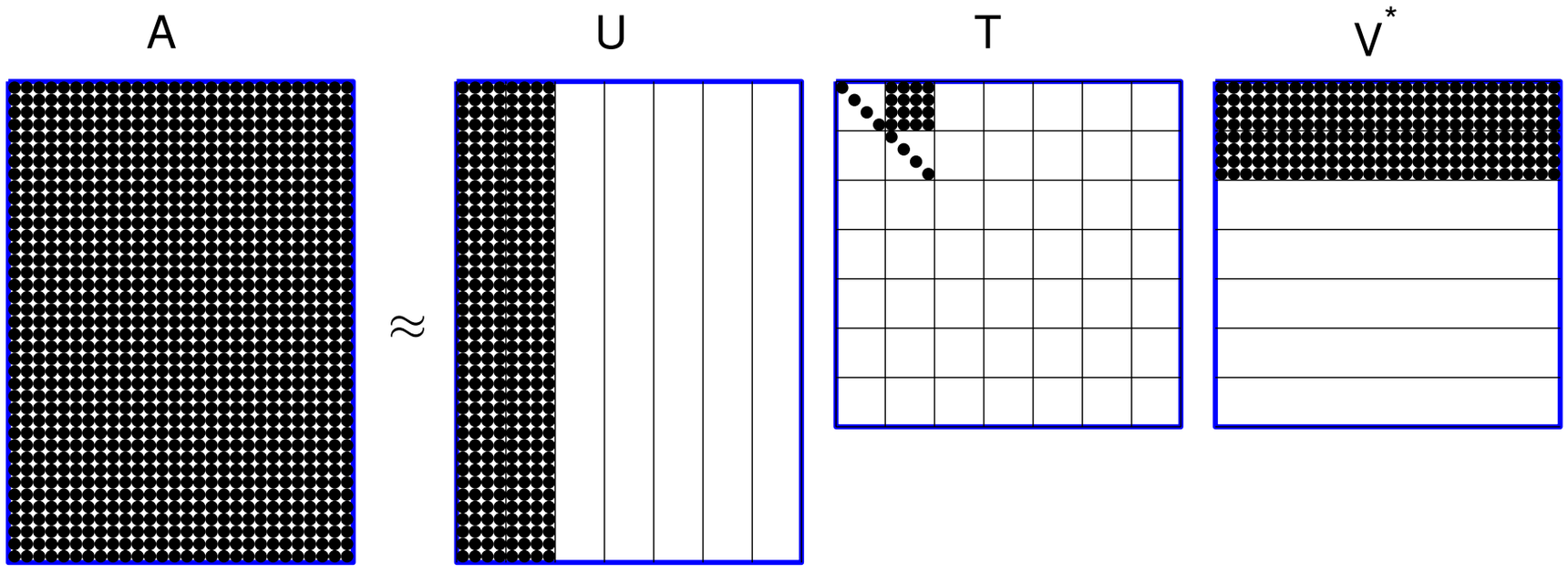
Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

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.

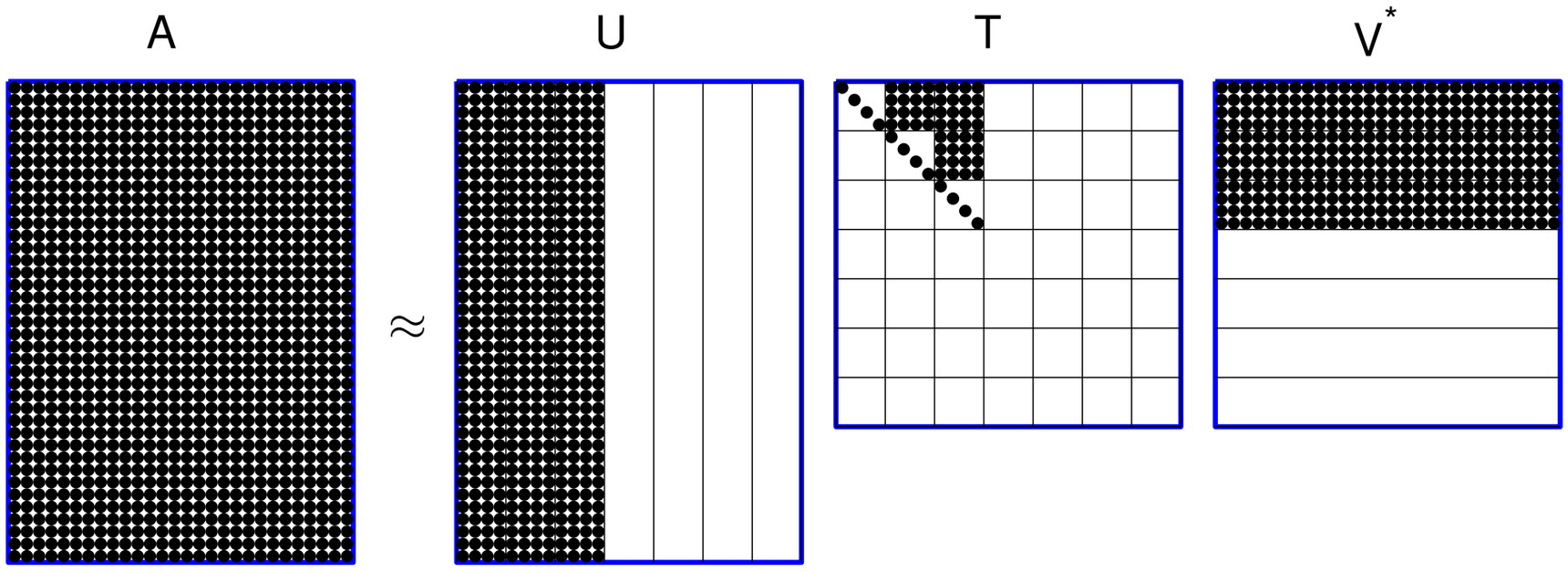
Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

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

Illustration for block size $b = 4$:



The rank-12 approximation.

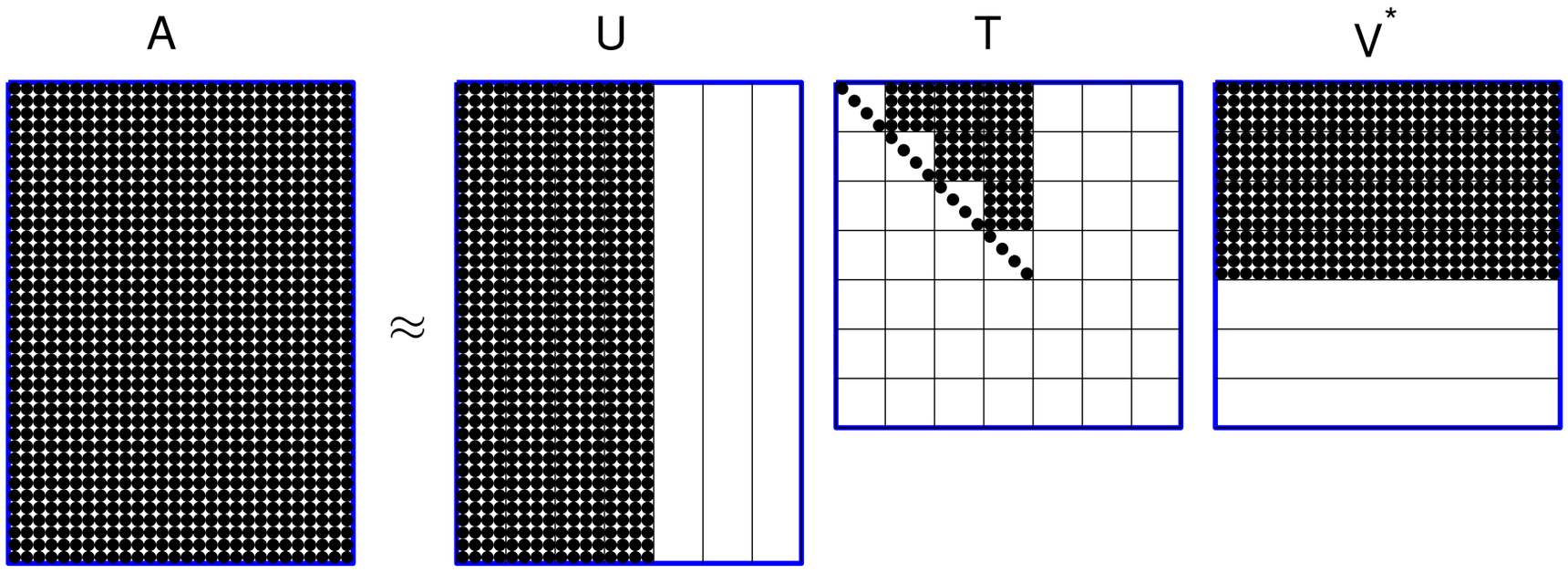
Recent work 2 (out of 5): Adaptive rank determination

Suppose that you seek to use the RSVD to compute a low rank approximation to a given matrix **A**, but you do not know its numerical rank in advance.

How do you proceed?

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

Illustration for block size $b = 4$:



The rank-16 approximation.

Recent work 2 (out of 5): Adaptive rank determination

Numerical experiments illustrating how close the UTV is to the SVD

As a consequence of the fact that the super-diagonal elements of **T** are very small, the diagonal elements of **T** are excellent approximants to the singular values of **A**:

$$\mathbf{T}(j,j) \approx \sigma_j, \qquad j = 1,\, 2\, \ldots,\, \min(m,n).$$

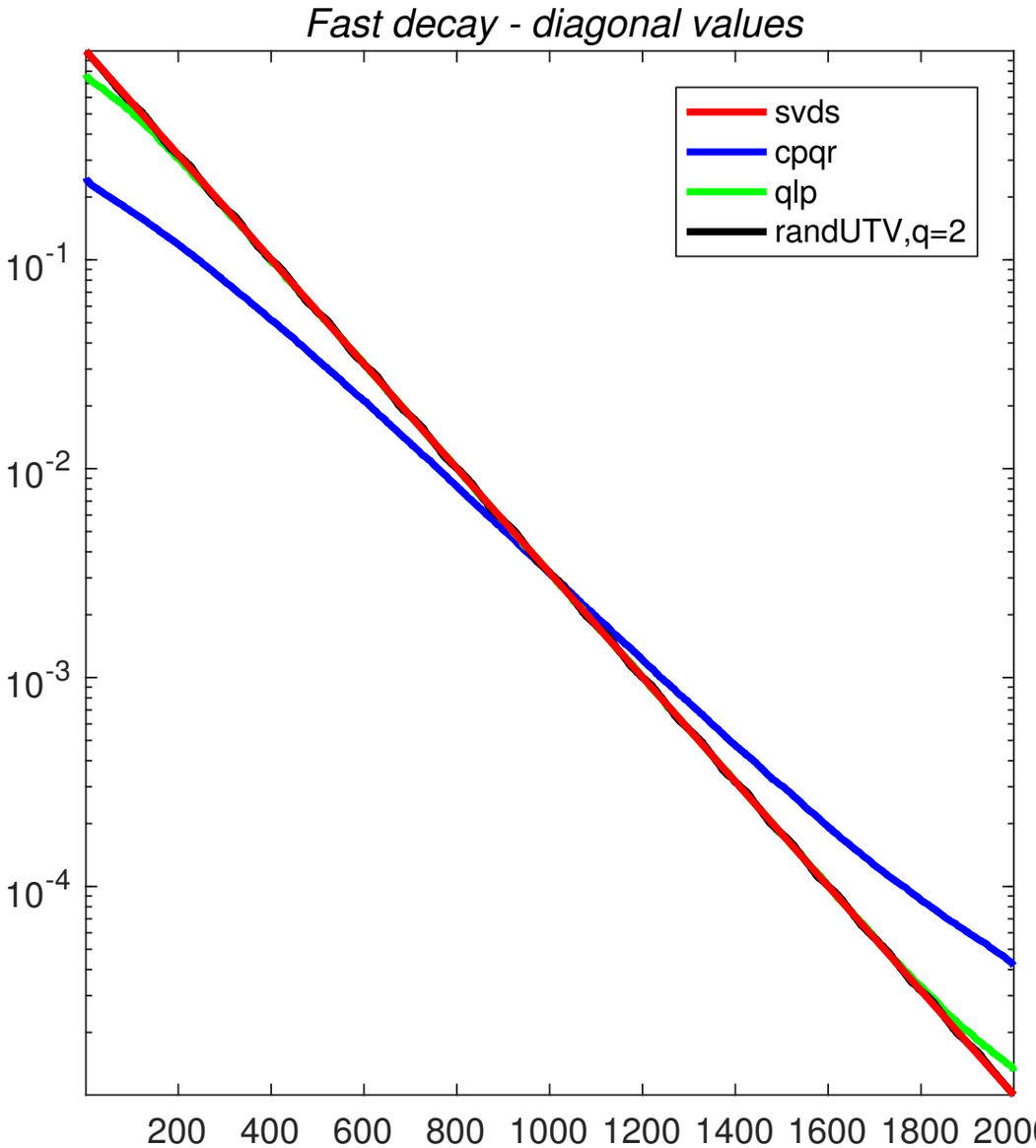
Recent work 2 (out of 5): Adaptive rank determination

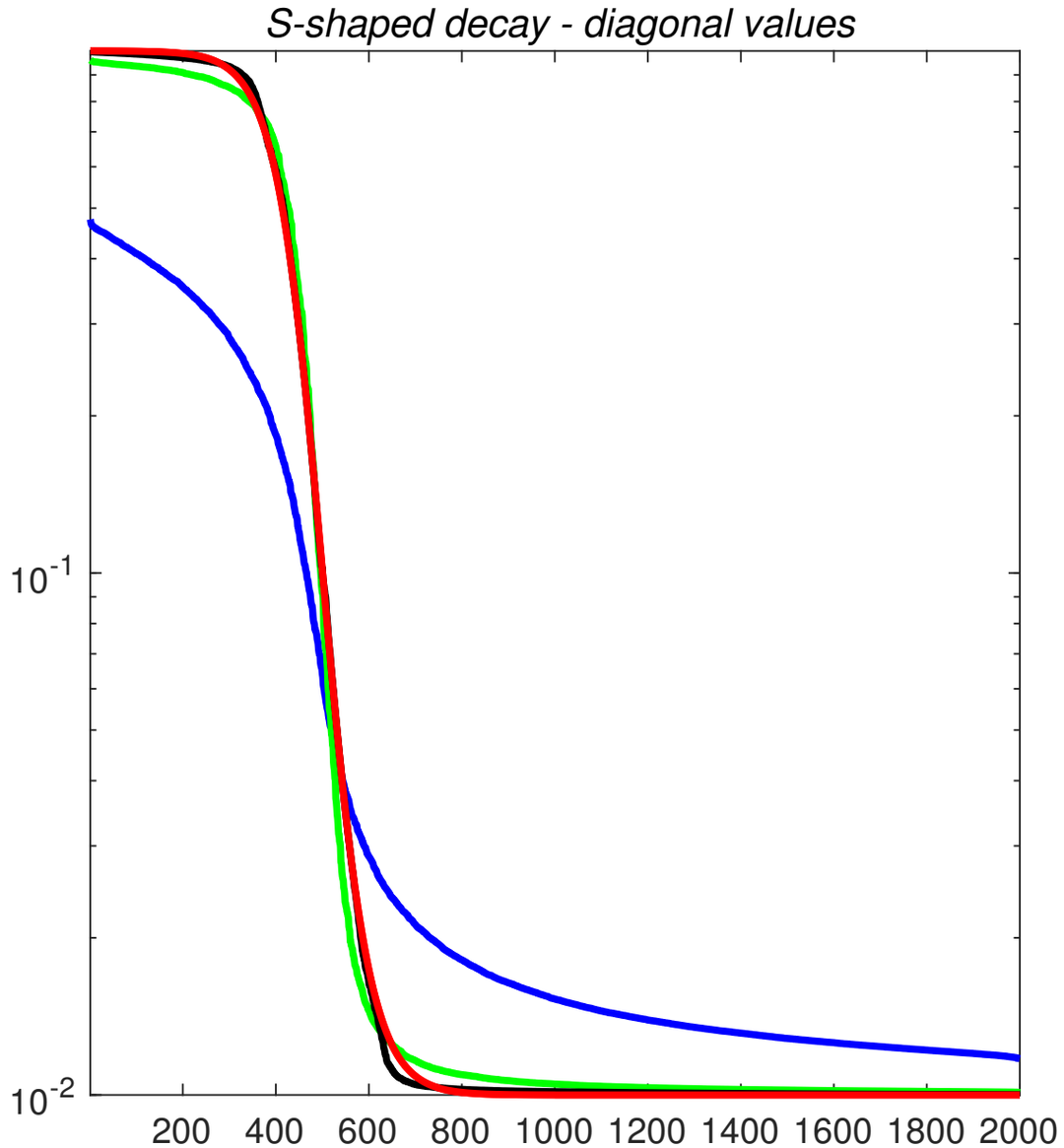
Numerical experiments illustrating how close the UTV is to the SVD

As a consequence of the fact that the super-diagonal elements of **T** are very small, the diagonal elements of **T** are excellent approximants to the singular values of **A**:

$$\mathbf{T}(j,j) \approx \sigma_j, \qquad j = 1, 2 \ldots, \min(m,n).$$

Numerical illustration:





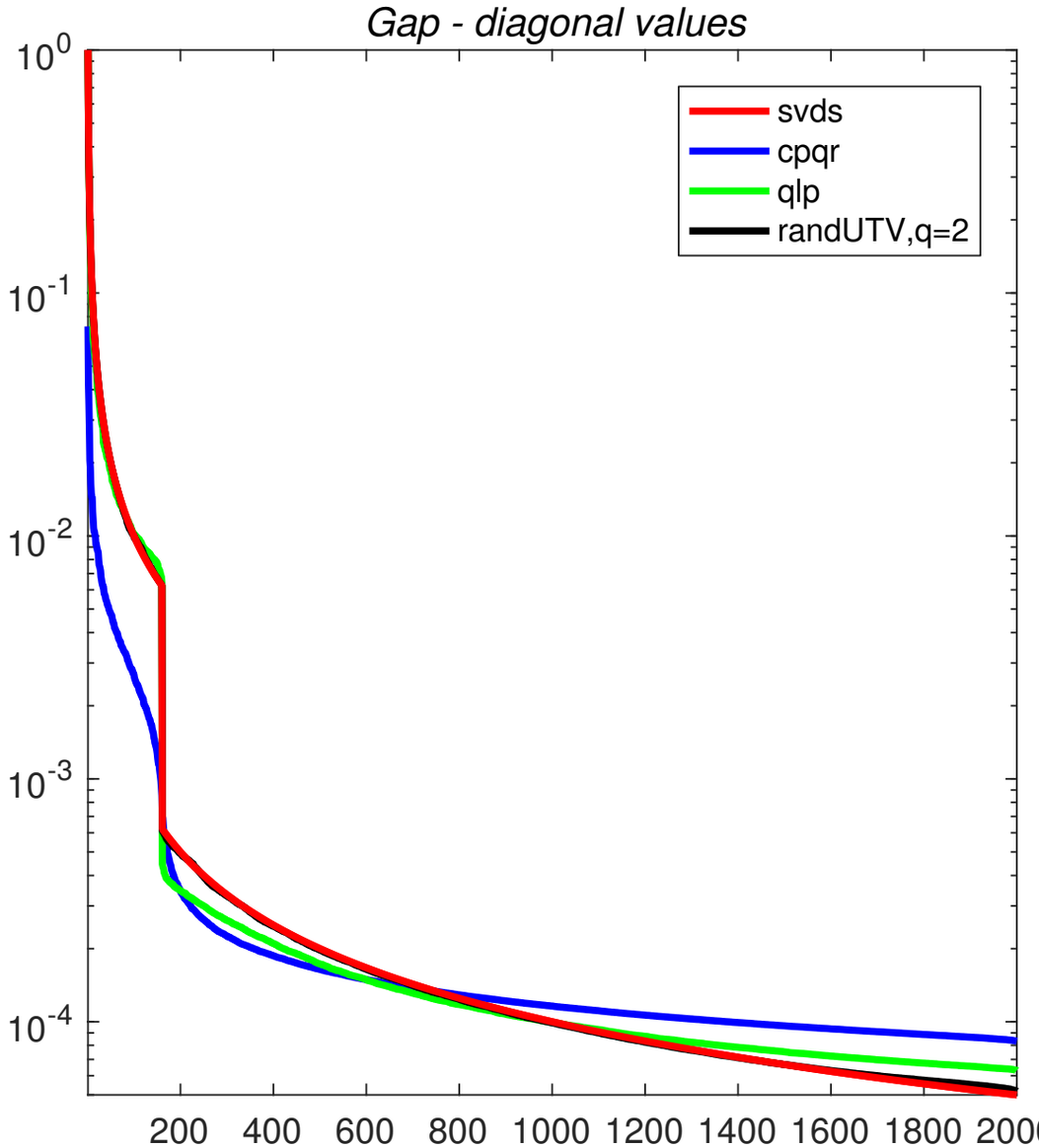
Recent work 2 (out of 5): Adaptive rank determination

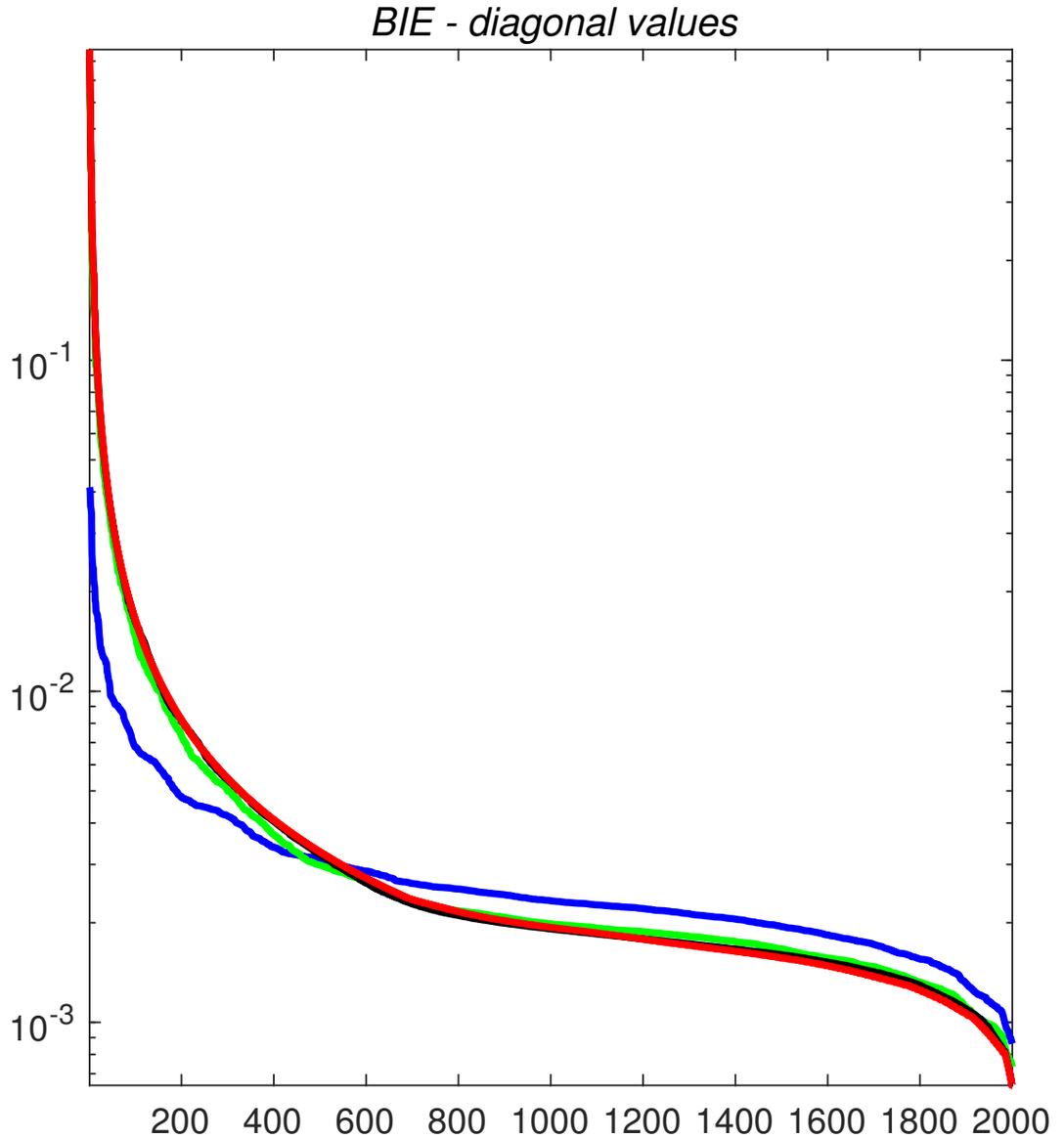
Numerical experiments illustrating how close the UTV is to the SVD

As a consequence of the fact that the super-diagonal elements of **T** are very small, the diagonal elements of **T** are excellent approximants to the singular values of **A**:

$$\mathbf{T}(j,j) \approx \sigma_j, \qquad j = 1, 2 \ldots, \min(m,n).$$

Numerical illustration:





Recent work 2 (out of 5): Adaptive rank determination

Numerical experiments illustrating how close the UTV is to the SVD

As a consequence of the fact that the super-diagonal elements of **T** are very small, the diagonal elements of **T** are excellent approximants to the singular values of **A**:

$$\mathbf{T}(j,j) \approx \sigma_j, \qquad j = 1, 2 \ldots, \min(m,n).$$

Observation: You now have a basically perfect stopping criterion!

Note in particular that it is accurate with respect to *operator norm!* (Not just Frobenius.)

Recent work 3 (out of 5): Downdating of a small sketch

It turns out that you actually do not have to redraw a random sample from the matrix at every step — *you extract a single small sample, and then just recycle it!*

Recent work 3 (out of 5): Downdating of a small sketch

It turns out that you actually do not have to redraw a random sample from the matrix at every step — *you extract a single small sample, and then just recycle it!*

In randUTV, the first step is to draw a sample from $\mathbf{A}$:

$$\mathbf{Y}_1 = \mathbf{A}\Omega_1.$$

We use $\mathbf{Y}_1$ to build unitary $\mathbf{U}_1$ and $\mathbf{V}_1$ and form

$$\mathbf{A}^{(1)} := \mathbf{U}_1^* \mathbf{A} \mathbf{V}_1 = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{0} & \mathbf{T}_{22} \end{bmatrix}.$$

We next want a sample from $\mathbf{A}^{(1)}$. Instead of generating a new draw Ω_2 from a Gaussian distribution and setting $\mathbf{Y}_2 = \mathbf{A}^{(1)}\Omega_2$, we proceed as follows:

$$\mathbf{Y}_2 := \mathbf{U}_1^* \mathbf{Y}_1 = \mathbf{U}_1^* \mathbf{A} \Omega_1 = \mathbf{U}_1^* \mathbf{A} \mathbf{V}_1 \mathbf{V}_1^* \Omega_1 = \mathbf{A}^{(1)} \Omega_2$$

where $\Omega_2 := \mathbf{V}_1^* \Omega_1$.

Question: Does Ω_2 have a Gaussian distribution?

Recent work 3 (out of 5): Downdating of a small sketch

It turns out that you actually do not have to redraw a random sample from the matrix at every step — *you extract a single small sample, and then just recycle it!*

In randUTV, the first step is to draw a sample from $\mathbf{A}$:

$$\mathbf{Y}_1 = \mathbf{A}\Omega_1.$$

We use $\mathbf{Y}_1$ to build unitary $\mathbf{U}_1$ and $\mathbf{V}_1$ and form

$$\mathbf{A}^{(1)} := \mathbf{U}_1^* \mathbf{A} \mathbf{V}_1 = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{0} & \mathbf{T}_{22} \end{bmatrix}.$$

We next want a sample from $\mathbf{A}^{(1)}$. Instead of generating a new draw Ω_2 from a Gaussian distribution and setting $\mathbf{Y}_2 = \mathbf{A}^{(1)}\Omega_2$, we proceed as follows:

$$\mathbf{Y}_2 := \mathbf{U}_1^* \mathbf{Y}_1 = \mathbf{U}_1^* \mathbf{A} \Omega_1 = \mathbf{U}_1^* \mathbf{A} \mathbf{V}_1 \mathbf{V}_1^* \Omega_1 = \mathbf{A}^{(1)} \Omega_2$$

where $\Omega_2 := \mathbf{V}_1^* \Omega_1$.

Question: Does Ω_2 have a Gaussian distribution?

No. But in practice there is no discernible difference.

Recent work 4 (out of 5): Interpolatory and CUR decompositions.

Question: Suppose that $\mathbf{A}$ is an $m \times n$ matrix of rank k (exact, for now), and that you are given a matrix $\mathbf{Y}$ whose columns span the columns of $\mathbf{A}$. In other words,

$$\text{ran}(\mathbf{A}) \subseteq \text{ran}(\mathbf{C}).$$

How do you efficiently use $\mathbf{Y}$ to build a rank-k approximation to $\mathbf{A}$?

Method 1: (Standard, we used this in construction of RSVD) Observe that

$$\mathbf{A} = \mathbf{Y}\mathbf{Y}^{\dagger}\mathbf{A} = \{\text{Do QR factorization } \mathbf{Y} = \mathbf{Q}\mathbf{R}\} = \mathbf{Q}\mathbf{Q}^{*}\mathbf{A} = \{\text{Set } \mathbf{B} := \mathbf{Q}^{*}\mathbf{A}\} = \mathbf{Q}\mathbf{B}.$$

Cost is $O(mk^2)$ to build $\mathbf{Q}$ and $O(mnk)$ to form $\mathbf{B}$.

Method 2: (Faster)

Recent work 4 (out of 5): Interpolatory and CUR decompositions.

Question: Suppose that $\mathbf{A}$ is an $m \times n$ matrix of rank k (exact, for now), and that you are given a matrix $\mathbf{Y}$ whose columns span the columns of $\mathbf{A}$. In other words,

$$\text{ran}(\mathbf{A}) \subseteq \text{ran}(\mathbf{C}).$$

How do you efficiently use $\mathbf{Y}$ to build a rank-k approximation to $\mathbf{A}$?

Method 1: (Standard, we used this in construction of RSVD) Observe that

$$\mathbf{A} = \mathbf{Y}\mathbf{Y}^{\dagger}\mathbf{A} = \{\text{Do QR factorization } \mathbf{Y} = \mathbf{Q}\mathbf{R}\} = \mathbf{Q}\mathbf{Q}^{*}\mathbf{A} = \{\text{Set } \mathbf{B} := \mathbf{Q}^{*}\mathbf{A}\} = \mathbf{Q}\mathbf{B}.$$

Cost is $O(mk^2)$ to build $\mathbf{Q}$ and $O(mnk)$ to form $\mathbf{B}$.

Method 2: (Faster) Build an "interpolatory decomposition" of $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{X}\mathbf{Y}(I_{\mathrm{S}}, :),$$

where I_{S} identifies a subset of the rows, and where $\mathbf{X}$ is well-conditioned. Then, *automatically*, we have

$$\mathbf{A} = \mathbf{X}\mathbf{A}(I_{\mathrm{S}}, :).$$

Cost is $O(mk^2)$ to build $\mathbf{X}$ and $O(nk)$ to form $\mathbf{A}(I_{\mathrm{S}}, :)$.

Building the ID can be done via Gram-Schmidt on the rows. (So CPQR on $\mathbf{Y}^{*}$.)

When the rank is approximate, then Method 2 can results in slightly larger errors.

Recent work 4 (out of 5): Interpolatory and CUR decompositions.

Note: Randomized ID is cheap. It does not need to revisit the entire matrix!

Randomized SVD:

- Draw random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \mathbf{A\Omega}$.
- Form $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$.
- Form $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
- Compute SVD of small matrix: $[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \texttt{svd}(\mathbf{B})$.
- Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Then $\mathbf{A} \approx \mathbf{UDV}^*$.

Randomized ID:

- Draw random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \mathbf{A\Omega}$.
- Form a row-ID of $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{XY}(I, :)$. *Use, e.g., CPQR. (Or LUPP!)*
- Set $\mathbf{R} = \mathbf{A}(I, :)$.

Then *automatically* $\mathbf{A} \approx \mathbf{XR}$.

The only "postprocessing" is extracting the columns in $\mathbf{R}$.

Recent work 4 (out of 5): Interpolatory and CUR decompositions.

Note: Randomized ID is cheap. It does not need to revisit the entire matrix!

Randomized SVD:

- Draw random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \mathbf{A\Omega}$.
- Form $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$.
- Form $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
- Compute SVD of small matrix: $[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \texttt{svd}(\mathbf{B})$.
- Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Then $\mathbf{A} \approx \mathbf{UDV}^*$.

Randomized CUR:

- Draw random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \mathbf{A\Omega}$.
- Form a row-ID of $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{XY}(I, :)$. *Use, e.g., CPQR. (Or LUPP!)*
- Set $\mathbf{R} = \mathbf{A}(I, :)$.
- Form a column-ID of $\mathbf{R}$ so that $\mathbf{R} = \mathbf{R}(:, J)\mathbf{Z}$. *Use, e.g., CPQR. (Or LUPP!)*
- Set $\mathbf{U} = (\mathbf{A}(I, J))^{\dagger}_{\epsilon}$. *Some care required...*

Then $\mathbf{A} \approx \mathbf{CUR}$.

The only "postprocessing" is extracting the submatrices $\mathbf{C}$ *and* $\mathbf{R}$.

Recent work 5 (out of 5): Iterative CUR

Finally, we describe the algorithm `IterativeCUR` that given a tolerance ε incrementally builds a CUR decomposition

$$\mathbf{A} \approx \mathbf{CUR}$$

such that $\|\mathbf{A} - \mathbf{CUR}\|_{\mathrm{F}} \leq \varepsilon$.

The method has the following characteristics:

- 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 $\mathbf{A} - \mathbf{CUR}$.
- Downdating 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.

With N. Pritchard, T. Park, Y. Nakatsukasa, arxiv:2509.21963

Recent work 5 (out of 5): Iterative CUR

Generate a sketch matrix $\mathbf{\Omega} \in \mathbb{R}^{1.1b \times n}$.

$\mathbf{I} = [\,]$; $\mathbf{I} = [\,]$; $\mathbf{C} = [\,]$; $\mathbf{U} = [\,]$; $\mathbf{R} = [\,]$;

$\mathbf{Y}_0^{\mathrm{row}} = \mathbf{\Omega A}$.

$\rho_0 = \|\mathbf{Y}_0^{\mathrm{row}}\|_{\mathrm{F}} / \|\mathbf{A}\|_{\mathrm{F}}$.

for $k = 1, 2, 3, \ldots$

if $[\rho_{k-1} \leq \varepsilon]$ **then stop**

Compute column indices J_k using the sketch $\mathbf{Y}_k^{\mathrm{row}}$. *[LUPP recommended.]*

$\mathbf{Y}_k^{\mathrm{col}} = \mathbf{A}(:, J_k) - \mathbf{CUR}(:, J_k)$.

Compute row indices I_k using the sketch $\mathbf{Y}_k^{\mathrm{col}}$. *[CPQR recommended.]*

$J = [J, J_k]$ and $I = [I, I_k]$.

$\mathbf{C} = [\mathbf{C}, \mathbf{A}(:, J_k)]$ and $\mathbf{R} = [\mathbf{R};\ \mathbf{A}(I_k, :)]$.

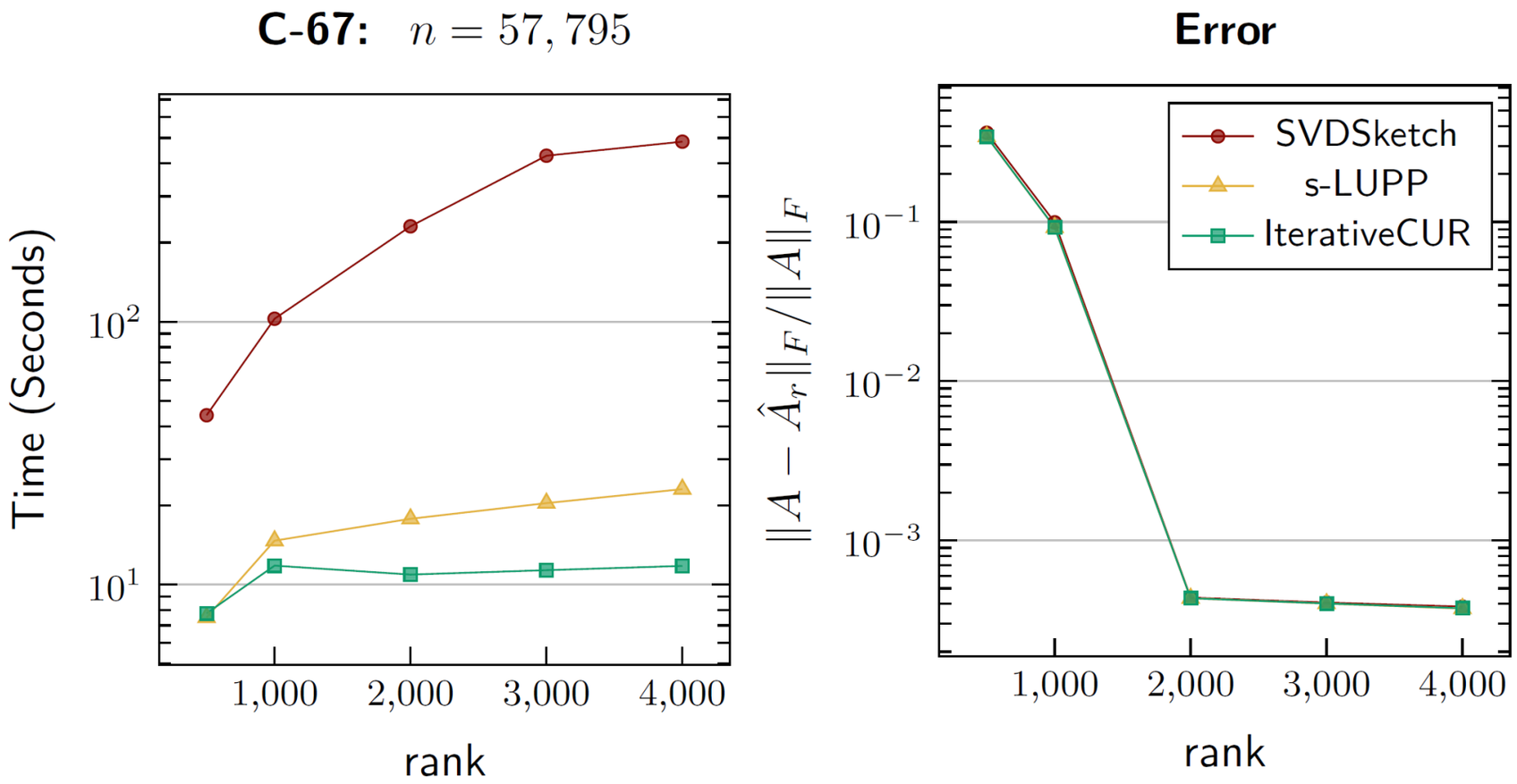
Compute $\mathbf{U}_k = \mathbf{A}(I, J)^{\dagger}$. *[Cheap Woodbury like update formula exists.]*

Compute $\mathbf{Y}_k^{\mathrm{row}} = \mathbf{\Omega A} - \mathbf{\Omega C}_k \mathbf{U}_k \mathbf{R}_k$. *[Downdating!]*

$\rho_k = \|\mathbf{Y}_k^{\mathrm{row}}\|_{\mathrm{F}} / \|\mathbf{A}\|_{\mathrm{F}}$.

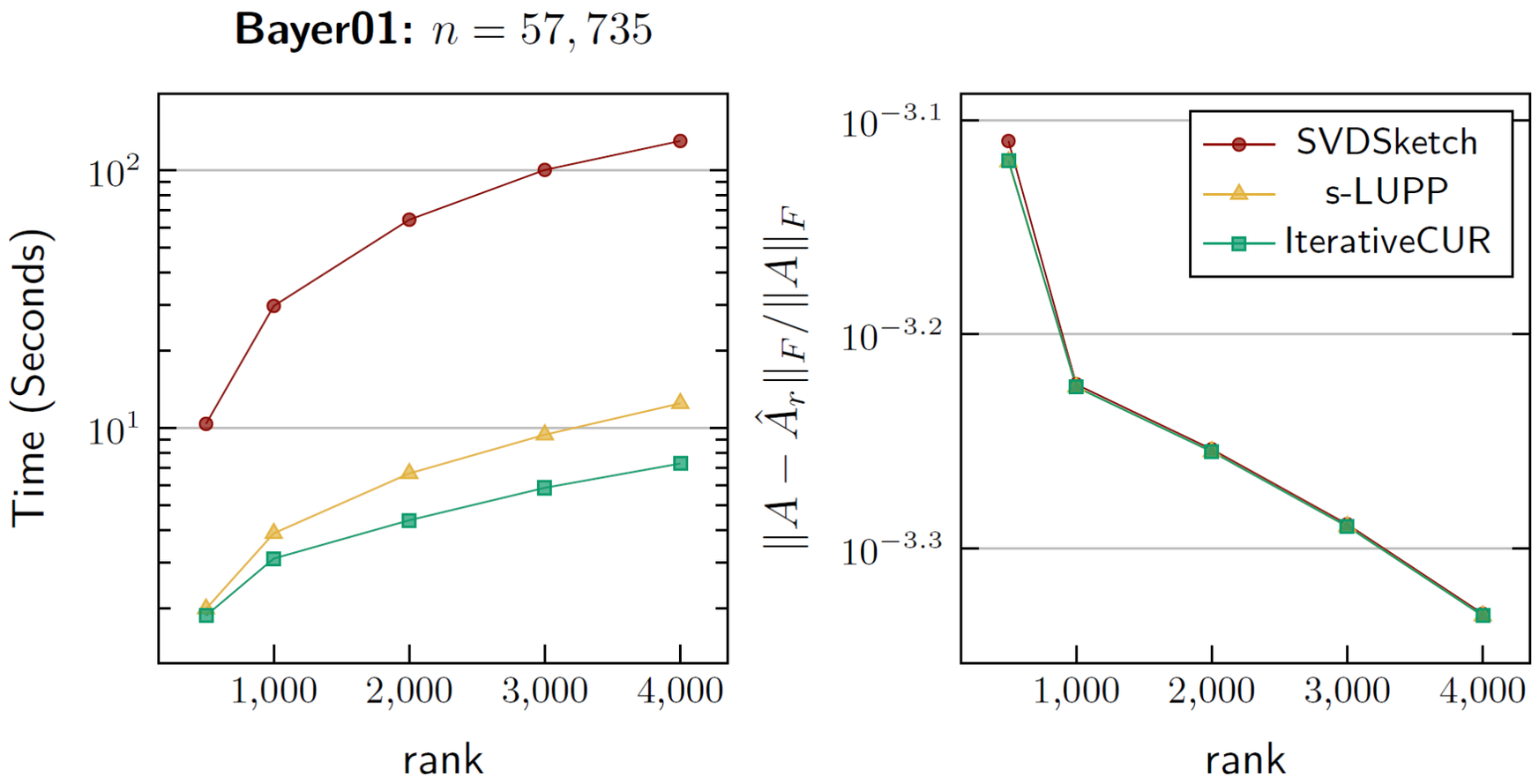
end for

Recent work 5 (out of 5): Iterative CUR



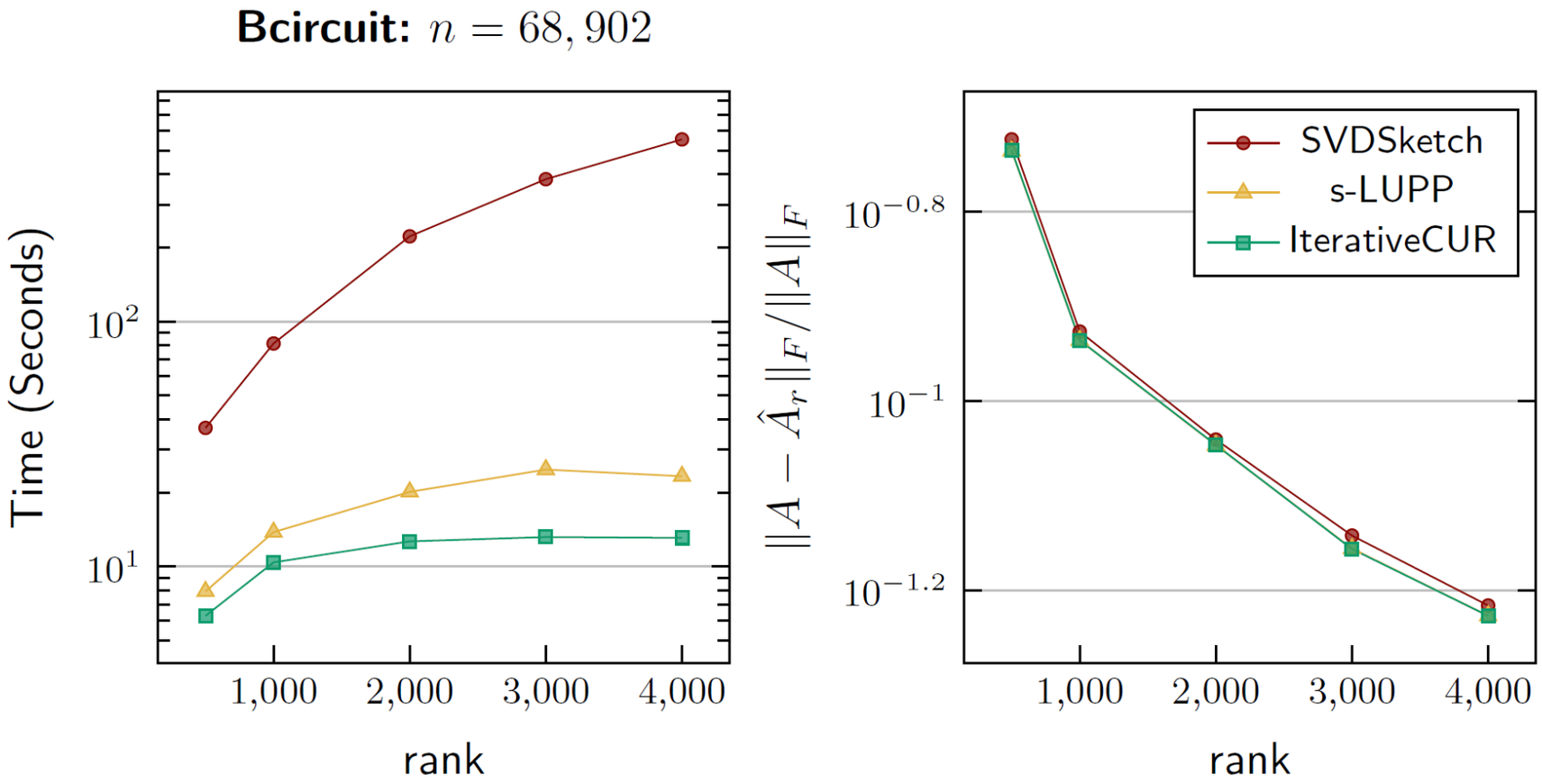
- SVDsketch: MATLAB's built-in [Yu-Yu-Li 18]
  - Roughly 'iterative RSVD' drawing new sketch each time
  - Cannot handle accuracy $\leq \epsilon_{\text{mach}}^{1/2} \approx 10^{-8}$
- s-LUPP: big sketch+LUPP
  - to test effect of reusing small sketch *G*
- IterativeCUR-LUPP: this work

Recent work 5 (out of 5): Iterative CUR



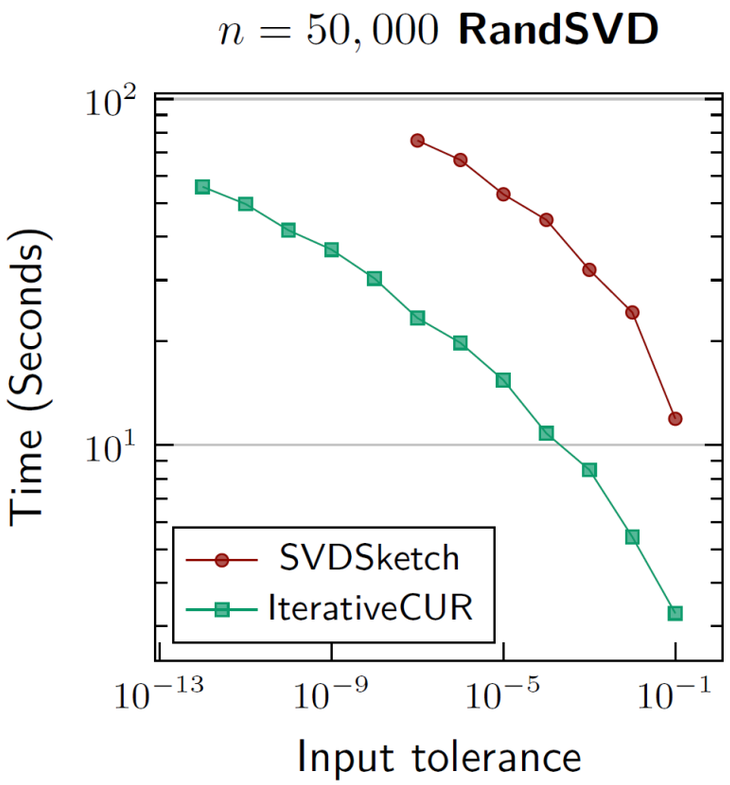
- SVDsketch: MATLAB's built-in [Yu-Yu-Li 18]
  - Roughly 'iterative RSVD' drawing new sketch each time
  - Cannot handle accuracy $\leq \epsilon_{\text{mach}}^{1/2} \approx 10^{-8}$
- s-LUPP: big sketch+LUPP
  - to test effect of reusing small sketch *G*
- IterativeCUR-LUPP: this work

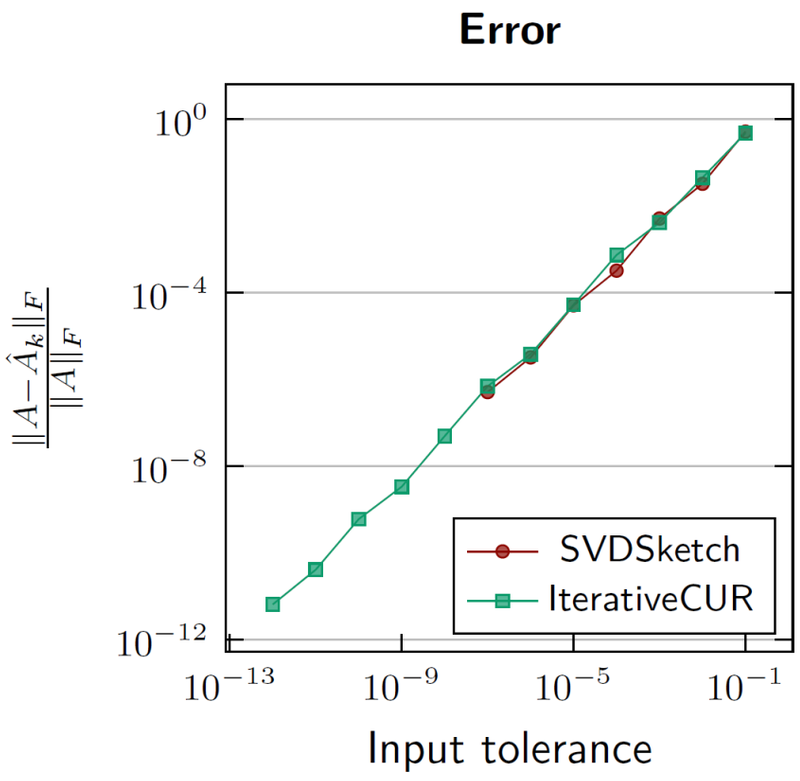
Recent work 5 (out of 5): Iterative CUR

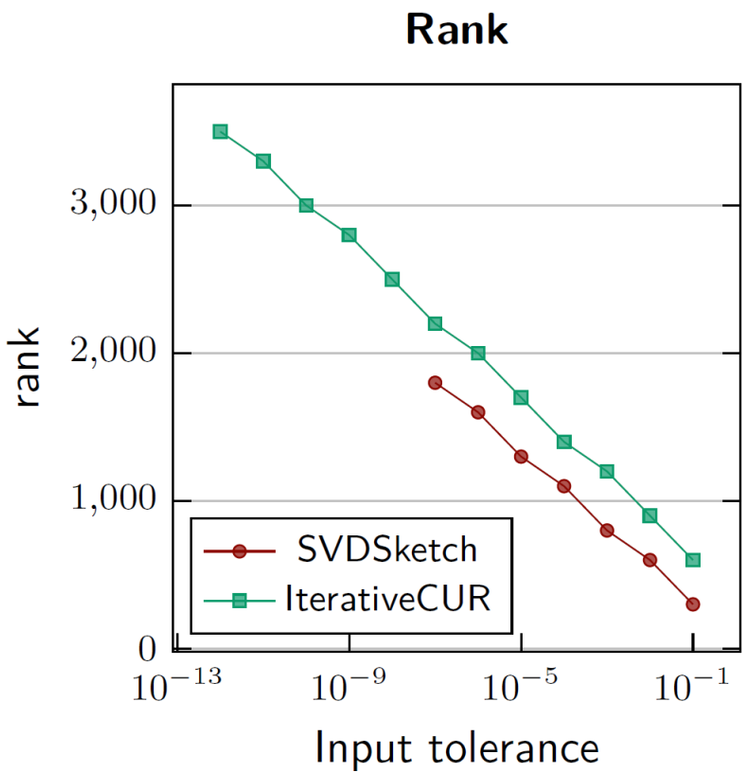


- SVDsketch: MATLAB's built-in [Yu-Yu-Li 18]
  - Roughly 'iterative RSVD' drawing new sketch each time
  - Cannot handle accuracy $\leq \epsilon_{\text{mach}}^{1/2} \approx 10^{-8}$
- s-LUPP: big sketch+LUPP
  - to test effect of reusing small sketch *G*
- IterativeCUR-LUPP: this work

Recent work 5 (out of 5): Iterative CUR

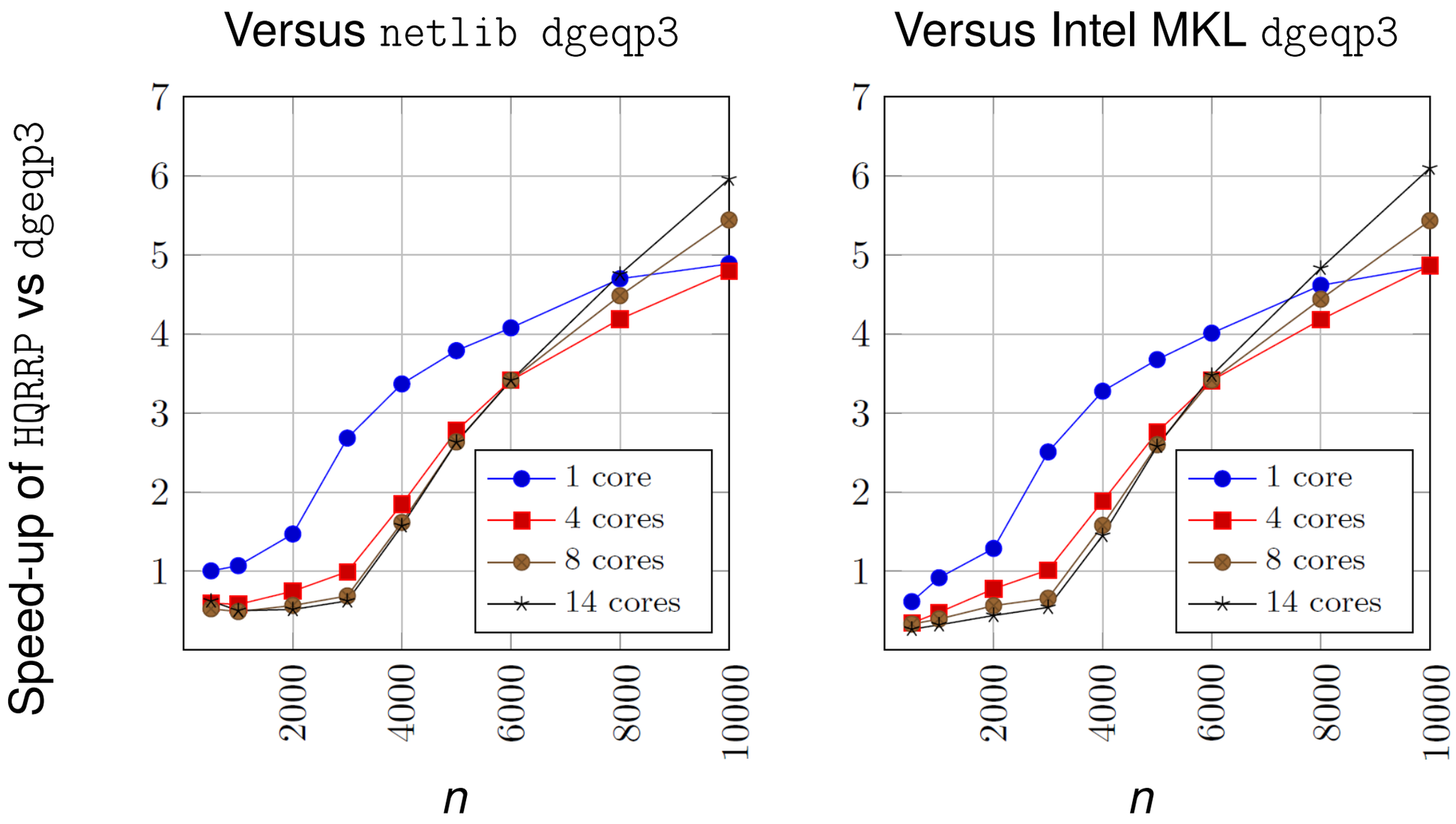






- SVDSketch fails for tolerance $\leq 10^{-8}$.
- Observe speedup attained by Iterative CUR.

Fun aside: Acceleration of *full* CPQR The randomized pivot selection technique can also be used to compute a column pivoted QR decomposition (CPQR). Resolves long standing issue of how to move flops from BLAS2 to BLAS3 in CPQR.



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/

Outline

- **Randomized dimension reduction – the basics** *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.

- **Recent advances**
  - A posteriori error estimation; rank adaptivity; sketch recycling.
  - Bring it all together: IterativeCUR

- **Sampling based methods for huge problems**
  - Monte Carlo style methods $\rightarrow$ less reliable, less accurate, less robust.
  - Enable the solution of stupendously large problems that would otherwise be intractable.
  - Resolved some long-standing open theoretical questions in linear algebra.

- **Randomized methods for differential and integral 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 $\{1, 2, \ldots, T\}$.

Draw an index $t \in \{1, 2, \ldots, T\}$ 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: $\bar{\mathbf{X}}_k = \frac{1}{k}\sum_{t=1}^{k} \mathbf{X}_i$ where $\mathbf{X}_t \sim \mathbf{X}$ are iid.

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

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. Resolution to problem of finding subsets of columns/rows with maximal *spanning volume*.

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. Resolution to problem of finding subsets of columns/rows with maximal *spanning volume*.

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 sym. pos. def. matrix. (Use diagonal elements to form sampling probabilities.) When combined with rejection sampling, sublinear complexity and high accuracy become possible [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

- **Randomized dimension reduction – the basics** *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.

- **Recent advances**
  - A posteriori error estimation; rank adaptivity; sketch recycling.
  - Bring it all together: IterativeCUR

- **Sampling based methods for huge problems**
  - Monte Carlo style methods $\rightarrow$ less reliable, less accurate, less robust.
  - Enable the solution of stupendously large problems that would otherwise be intractable.
  - Resolved some long-standing open theoretical questions in linear algebra.

- **Randomized methods for differential and integral 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

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

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:

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

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

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

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

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

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

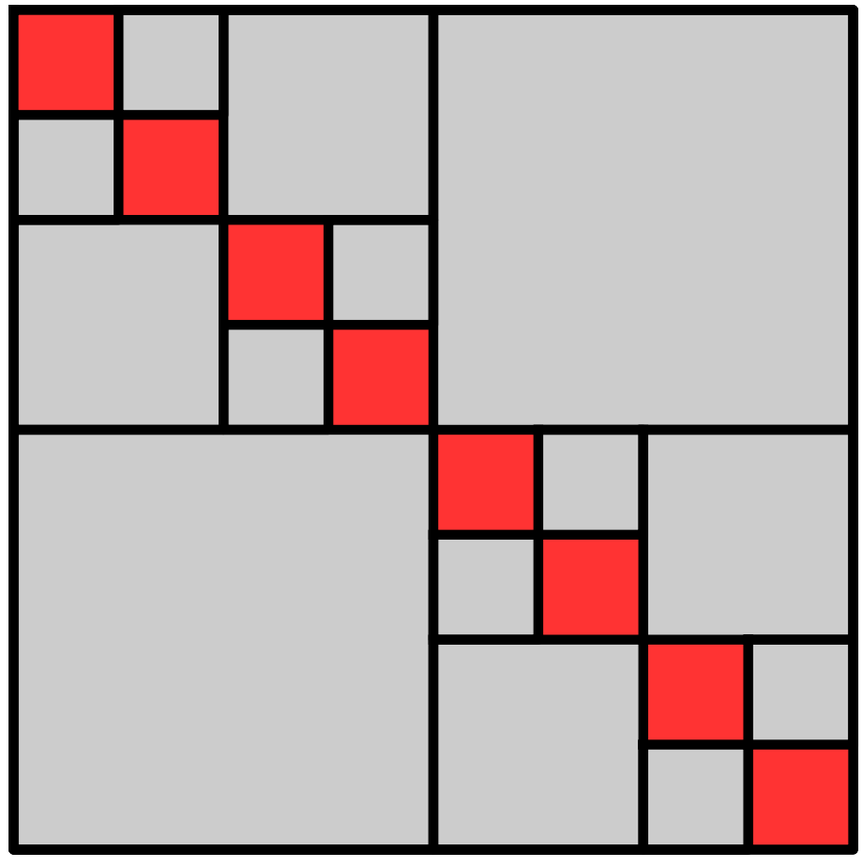
Consider a linear boundary value problem of the form

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

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

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

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 ...

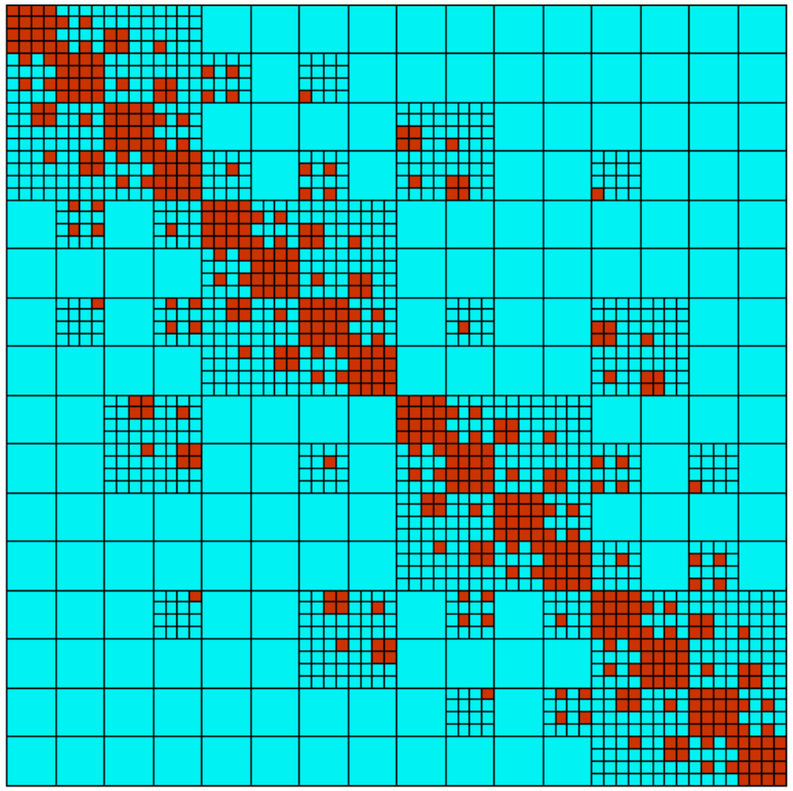


Image credit: Ambikasaran & Darve, arxiv.org #1407.1572

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

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

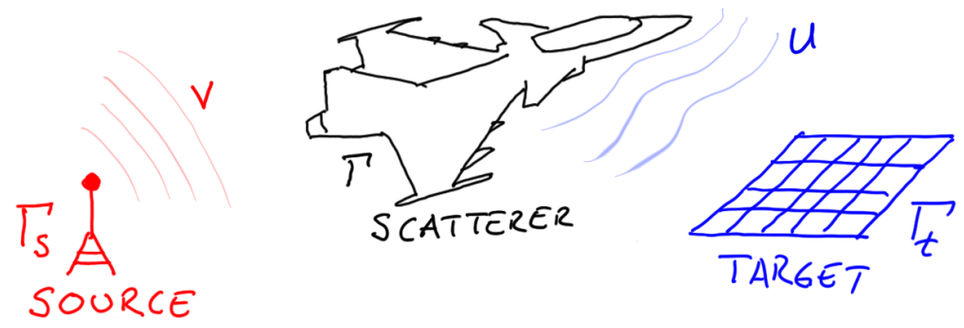
where Ω is a domain (2D or 3D) with boundary Γ, and where A is a linear elliptic differential operator; possibly with variable coefficients.

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: 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:



$$\text{(2)} \qquad \begin{cases} -\Delta u(\boldsymbol{x}) - \kappa^2 u(\boldsymbol{x}) = 0 & \boldsymbol{x} \in \mathbb{R}^3 \backslash \overline{\Omega} \\ u(\boldsymbol{x}) = v(\boldsymbol{x}) & \boldsymbol{x} \in \partial\Omega \\ \lim\limits_{|\boldsymbol{x}|\to\infty} |\boldsymbol{x}| \left(\partial_{|\boldsymbol{x}|} u(\boldsymbol{x}) - i\kappa\, u(\boldsymbol{x}) \right) = 0. \end{cases}$$

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

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

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 $\mathbb{R}^3 \backslash \overline{\Omega}$ is 3D.
- Equation (3) is inherently well-conditioned (as a "2^{nd} 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

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

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

$$T : f \mapsto n$$

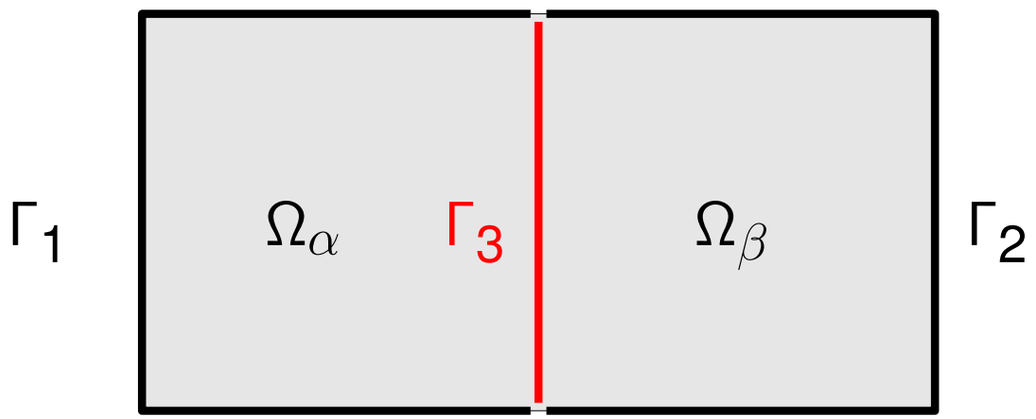
is known as the *Dirichlet-to-Neumann (DtN)* map.

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

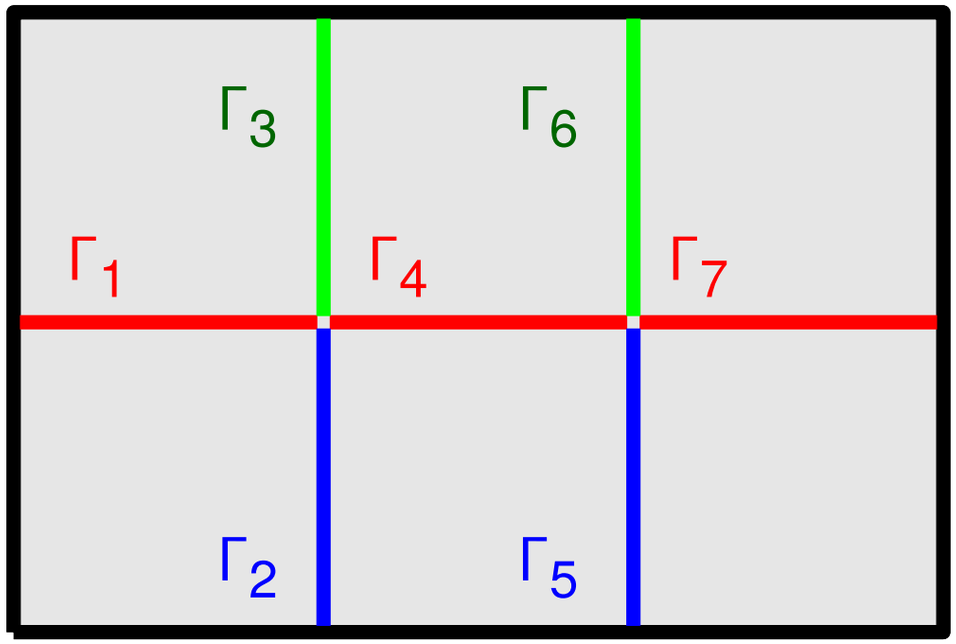
$$\begin{bmatrix} T^\alpha_{11} & T^\alpha_{13} \\ T^\alpha_{31} & T^\alpha_{33} \end{bmatrix} \begin{bmatrix} f_1 \\ f_3 \end{bmatrix} = \begin{bmatrix} n_1 \\ n^\alpha_3 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} T^\beta_{22} & T^\beta_{23} \\ T^\beta_{32} & T^\beta_{33} \end{bmatrix} \begin{bmatrix} f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} n_2 \\ n^\beta_3 \end{bmatrix}$$

where n^α_3 and n^β_3 represent the boundary fluxes through Γ_3 from Ω_α and Ω_β, respectively. Since $n^\alpha_3 + n^\beta_3 = 0$, we can now form an equation for the unknown quantity f_3 as

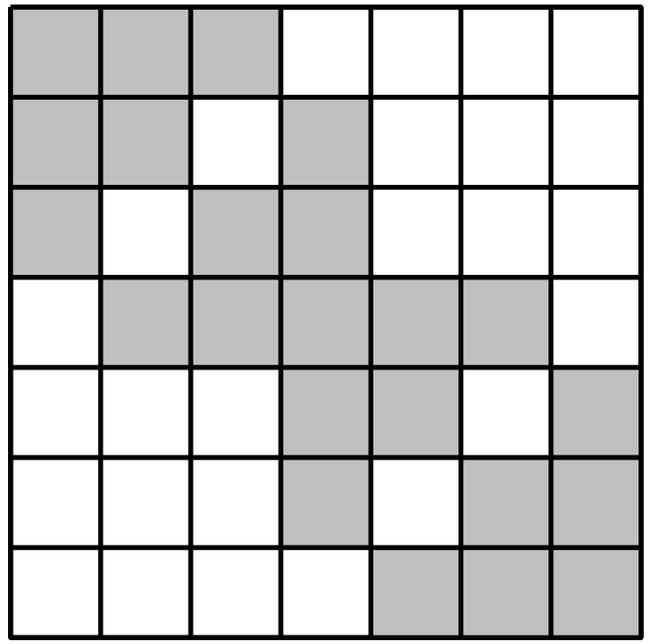
$$(T^\alpha_{33} + T^\beta_{33}) f_3 = -T^\alpha_{31} f_1 - T^\beta_{32} f_2.$$

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 $\{\Gamma_i\}_{i=1}^{7}$.

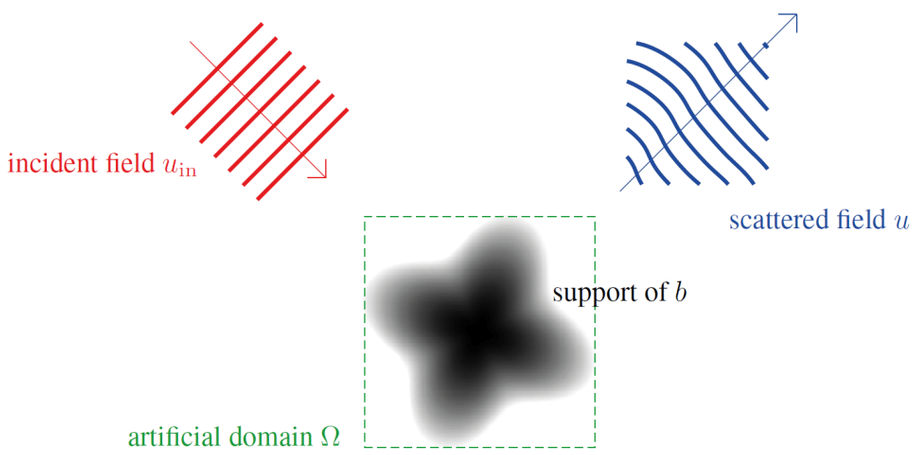
Example: Poincaré-Steklov operators (Dirichlet-to-Neumann, etc)

Consider the free space acoustic scattering problem

$$\begin{cases} -\Delta u(\boldsymbol{x}) - \kappa^2 (1 - b(\boldsymbol{x}))\, u(\boldsymbol{x}) = -\kappa^2 b(\boldsymbol{x})\, v(\boldsymbol{x}), & \boldsymbol{x} \in \mathbb{R}^2 \\ \lim\limits_{|\boldsymbol{x}|\to\infty} \sqrt{|\boldsymbol{x}|}\big(\partial_{|\boldsymbol{x}|} u(\boldsymbol{x}) - i\kappa\, u(\boldsymbol{x})\big) = 0, \end{cases}$$

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 support(b) $\subseteq \Omega$.

On Ω:	On Ω^{c}:
• Variable coefficient PDE.	• Constant coefficient PDE.
• Discretize the PDE.	• Use BIE.
• Build DtN for $\partial\Omega$.	• Build DtN for $\partial\Omega^{\text{c}}$.

Glue the domains together using the DtNs.

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

[Gillman, Barnett, Martinsson, 2015]

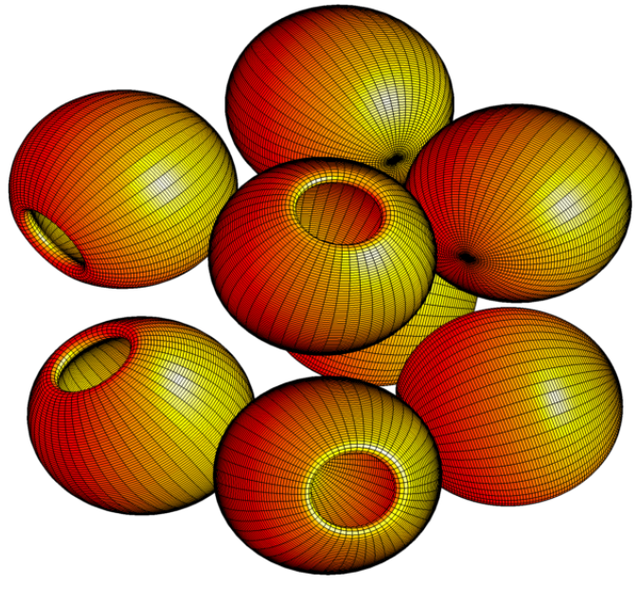
Example: Scattering operators

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 parallelizeable.
- Form a global system that uses the scattering matrices to describe how the bodies talk to each.

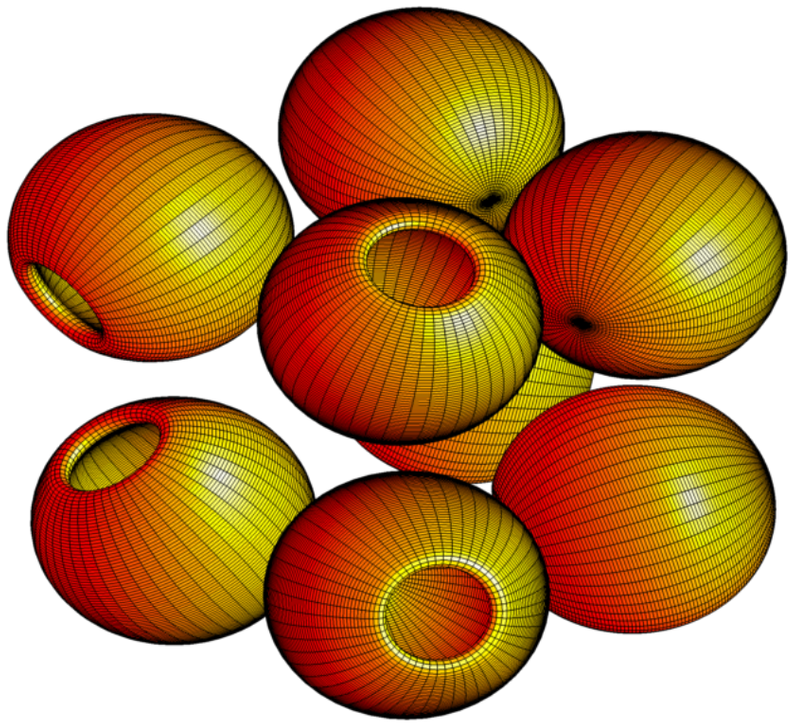
The benefit is that the global system you form this way is *far smaller* than a global system that fully resolves all individual scatters. (And often better conditioned too!)



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.



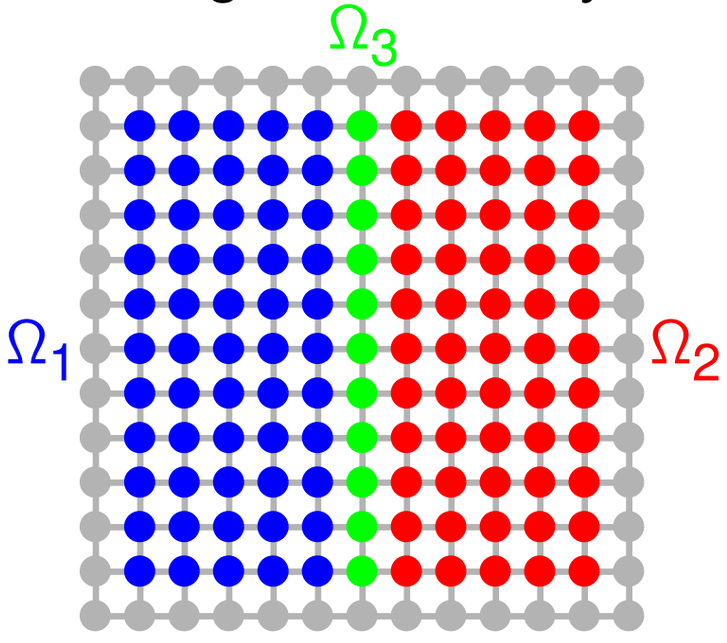
A hybrid direct/iterative solver is used (a highly accurate scattering matrix $\mathbf{S}_i$ is computed for body i) to form a global system

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

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.

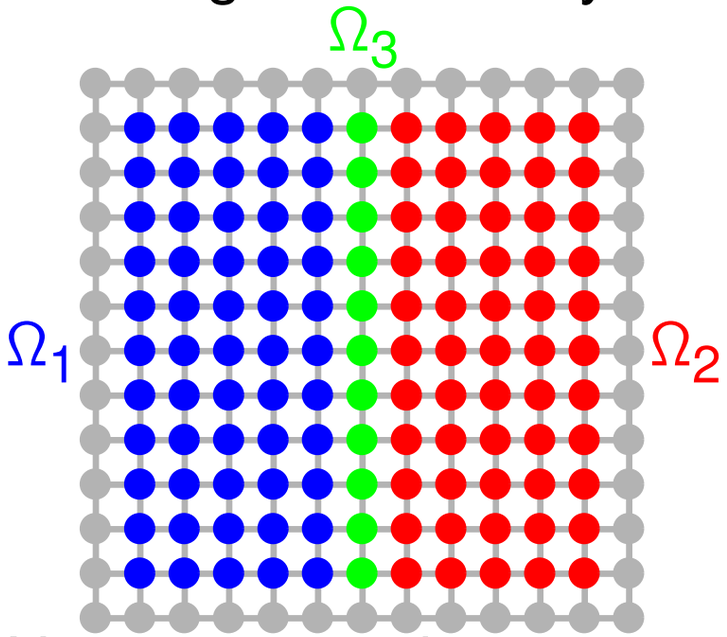
[2015, CAMWA, Hao/M./Young]

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



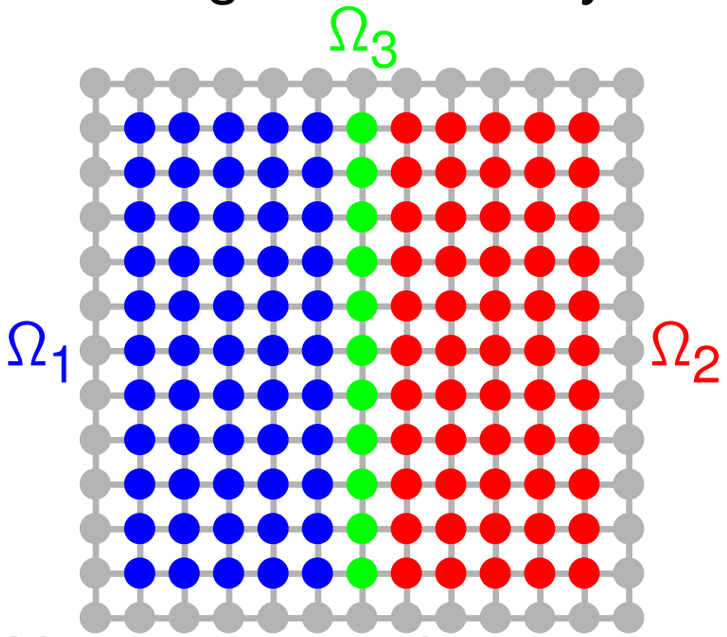
$$\mathbf{A} = \begin{array}{|c|c|c|}\hline \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \\ \hline \end{array}$$

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

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

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 $\mathbf{Au} = \mathbf{b}$. Let us partition the nodes into three sets as follows:



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

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

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

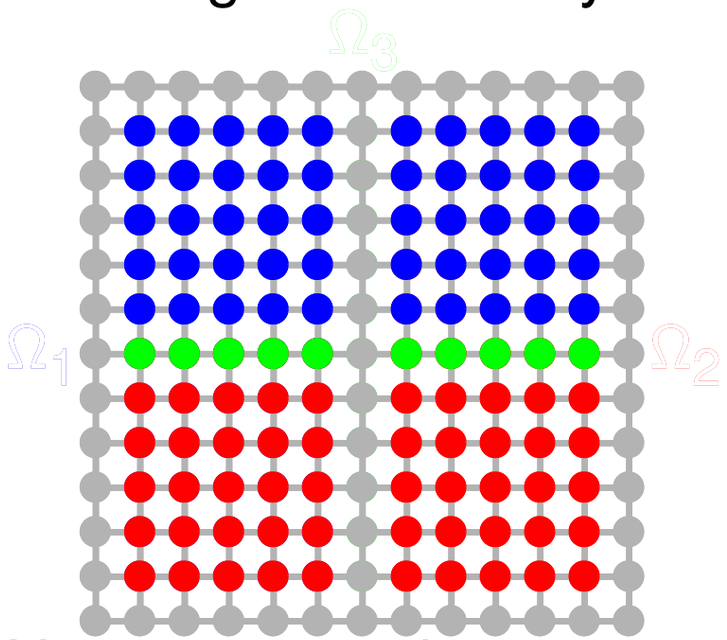
where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement.*

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

- Invert $\mathbf{A}_{11}$ to form $\mathbf{A}_{11}^{-1}$. $\quad size \sim N/2 \times N/2$
- Invert $\mathbf{A}_{22}$ to form $\mathbf{A}_{22}^{-1}$. $\quad size \sim N/2 \times N/2$
- Invert $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$. $\quad size \sim \sqrt{N} \times \sqrt{N}$

Notice the obvious recursion!

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



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

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 $\mathbf{A}_{11}^{-1}$. $\quad size \sim N/2 \times N/2$
- Invert $\mathbf{A}_{22}$ to form $\mathbf{A}_{22}^{-1}$. $\quad size \sim N/2 \times N/2$
- Invert $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$. $\quad size \sim \sqrt{N} \times \sqrt{N}$

Notice the obvious recursion!

Example: Schur complements

Typically, nested dissection orderings are more complicated:

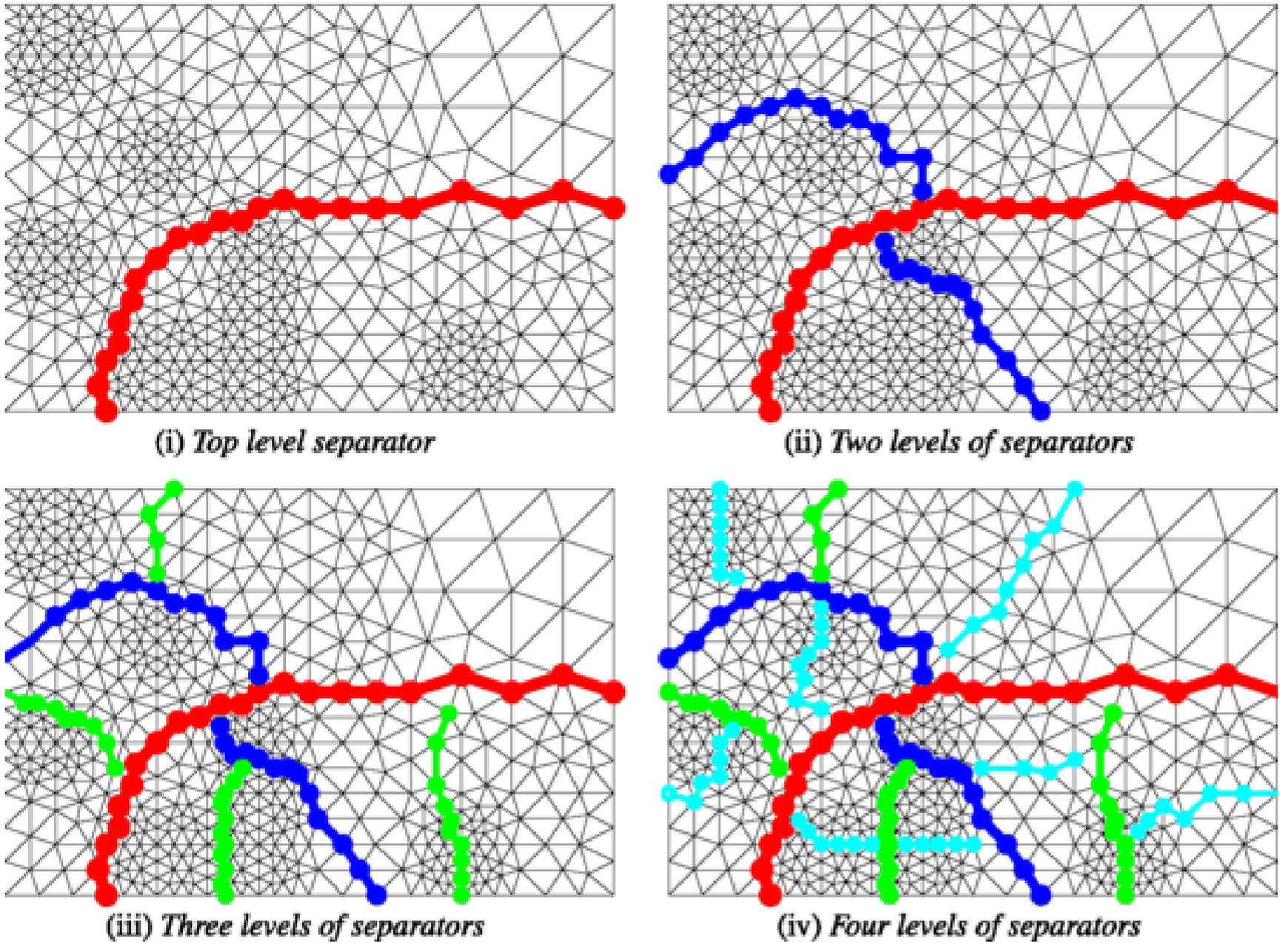


Image credit: Jianlin Xia, "Robust and Efficient Multifrontal Solver for Large Discretized PDEs", 2012

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 discretizated 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:

	Build stage		*Solve stage*	
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 $N \to \infty$. 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:

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

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

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

The kernels are derived from the corresponding fundamental solutions:

$$\begin{aligned} s(\boldsymbol{x},\boldsymbol{y}) &= \phi(\boldsymbol{x}-\boldsymbol{y}),\\ d(\boldsymbol{x},\boldsymbol{y}) &= \partial_{\boldsymbol{n}(\boldsymbol{y})}\phi(\boldsymbol{x}-\boldsymbol{y}),\\ s_\kappa(\boldsymbol{x},\boldsymbol{y}) &= \phi_\kappa(\boldsymbol{x}-\boldsymbol{y}),\\ d_\kappa(\boldsymbol{x},\boldsymbol{y}) &= \partial_{\boldsymbol{n}(\boldsymbol{y})}\phi_\kappa(\boldsymbol{x}-\boldsymbol{y}), \end{aligned}$$

where, as before,

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

Interaction ranks: Boundary integral equations

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

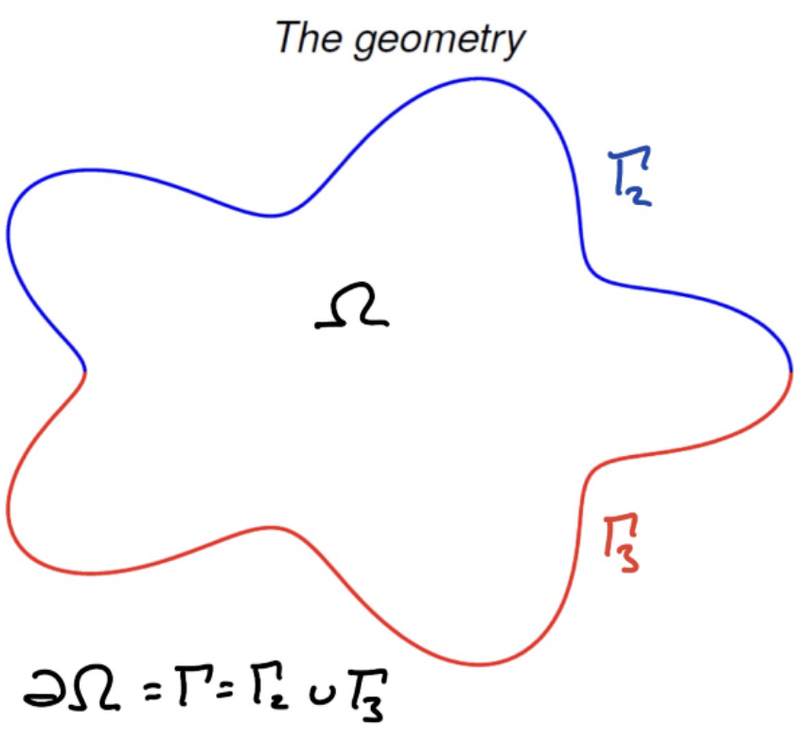
The first is a reformulation of a Dirichlet problem involving the Laplace equation:

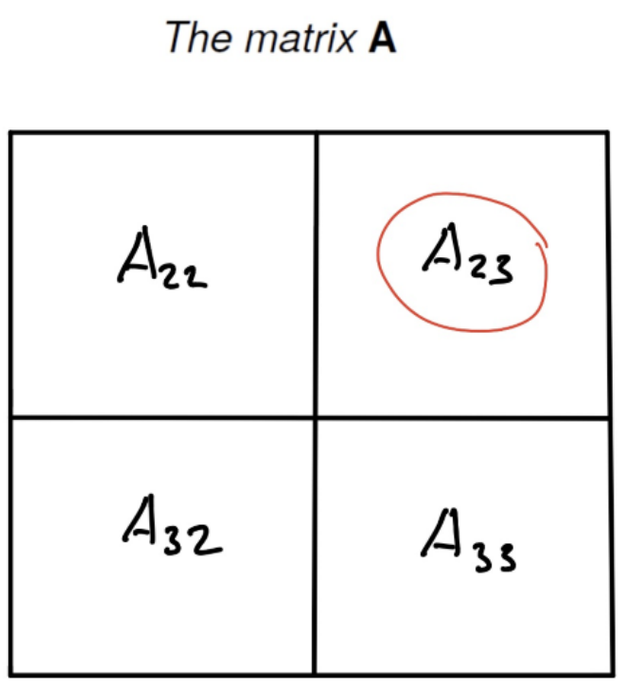
$$\alpha\sigma(\boldsymbol{x}) + \int_\Gamma \big(d(\boldsymbol{x},\boldsymbol{y}) + s(\boldsymbol{x},\boldsymbol{y})\big)\,\sigma(\boldsymbol{y})\,ds(\boldsymbol{y}) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma.$$

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

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

Let **A** denote the matrix resulting from discretization of either BIE.

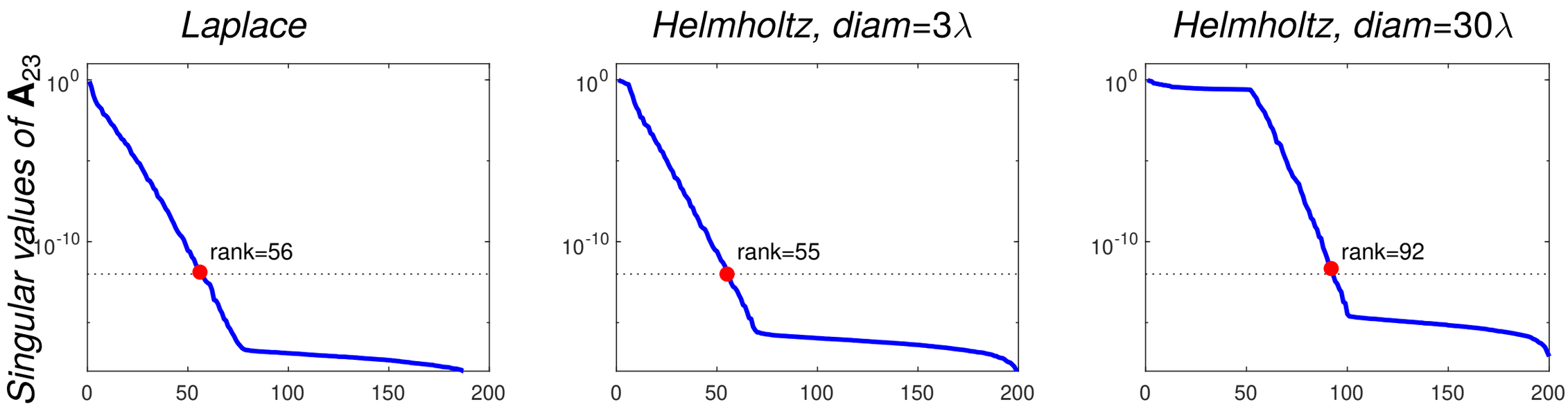




On the next slide, we show the singular values of the off-diagonal block $\mathbf{A}_{23}$.

Interaction ranks: Boundary integral equations

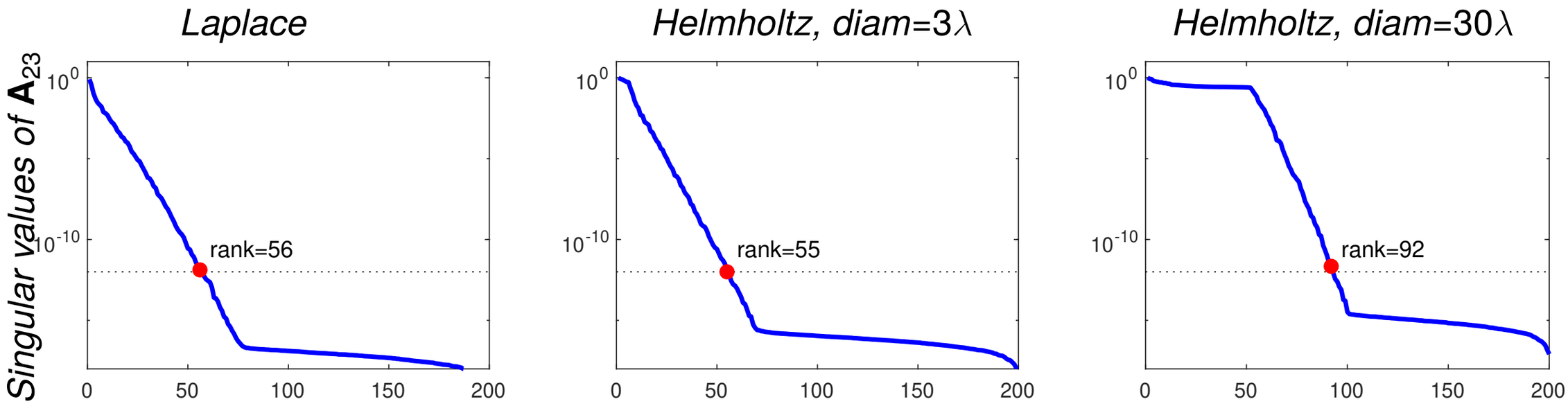
The ranks of an off-diagonal block of **A**:



This is all as expected. Somewhat accessible by analysis.

Interaction ranks: Boundary integral equations

The ranks of an off-diagonal block of **A**:

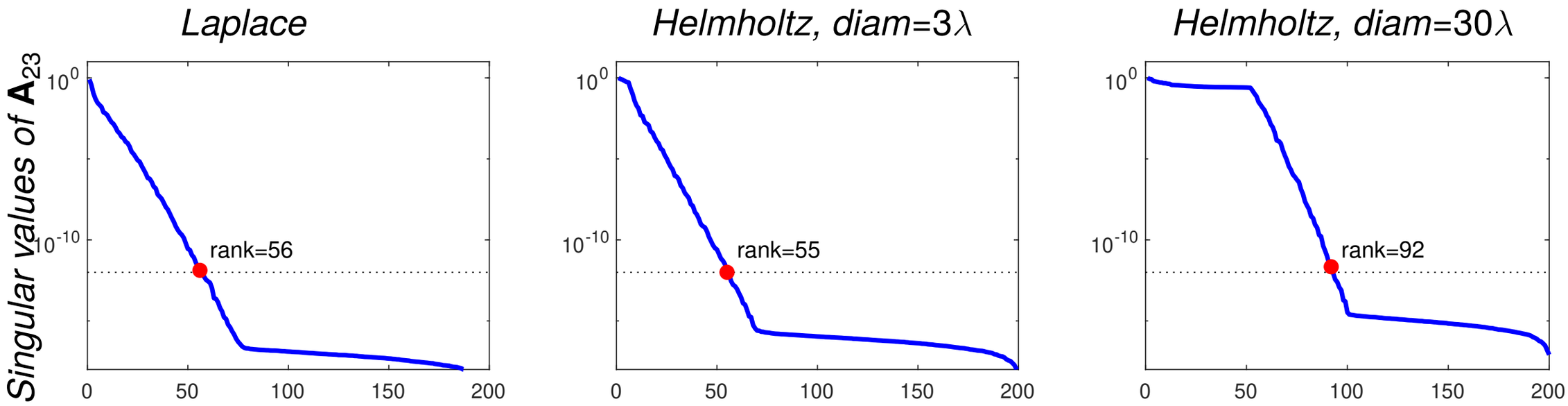


This is all as expected. Somewhat accessible by analysis.

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

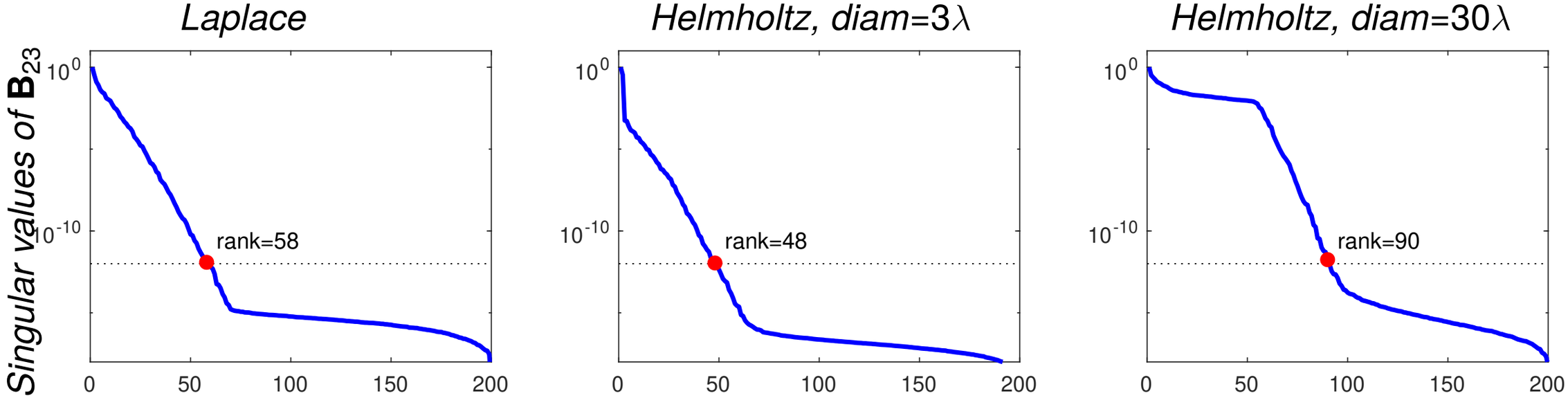
Interaction ranks: Boundary integral equations

The ranks of an off-diagonal block of **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 $\mathbf{B}_{23}$.

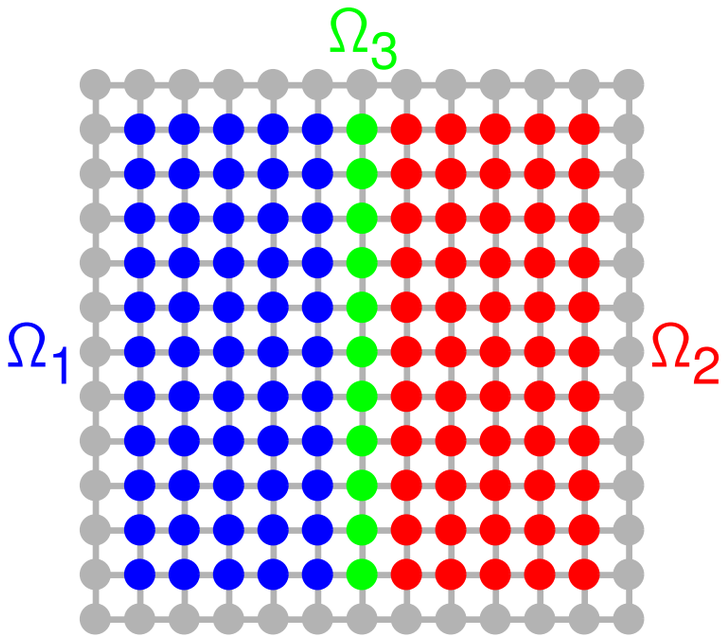


Remarkable similarity!

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

Interaction ranks: Stiffness matrix from finite difference discretization

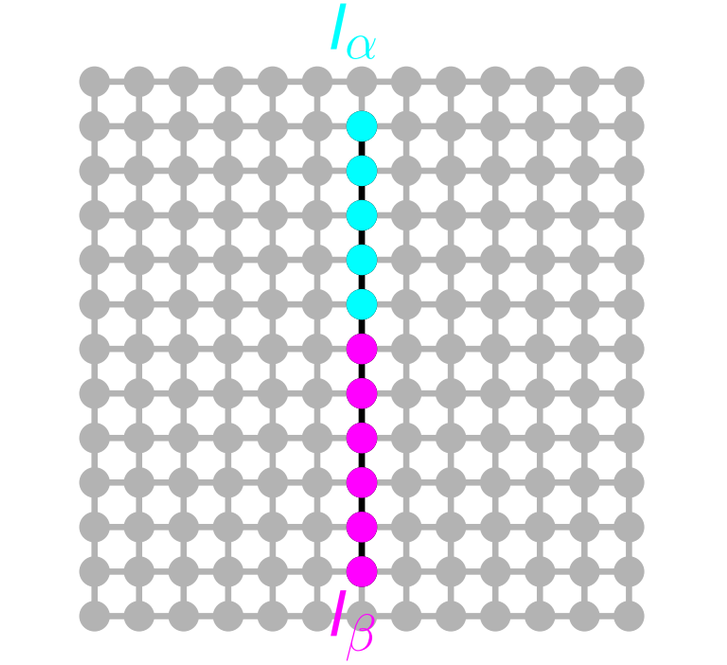
Recall our example of Laplace's equation discretized using the 5-point stencil.



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

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:

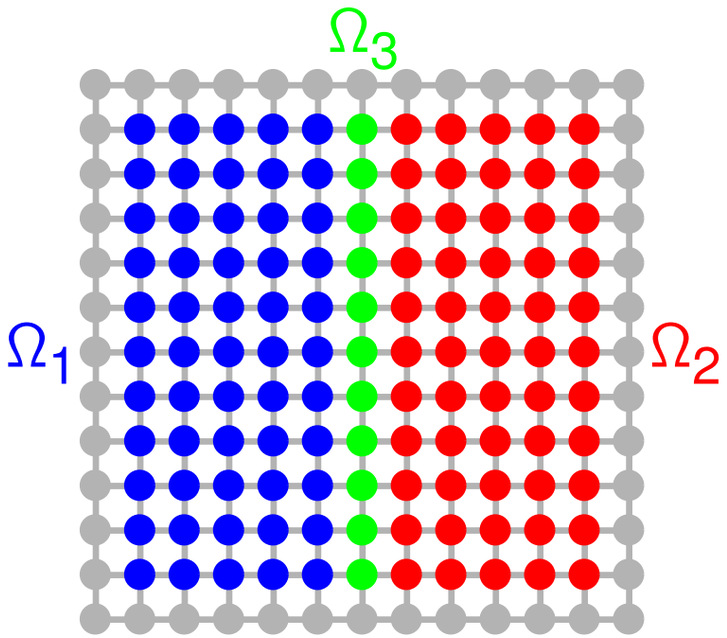


$$\mathbf{S} = \begin{array}{|c|c|} \hline \mathbf{S}_{\alpha\alpha} & \mathbf{S}_{\alpha\beta} \\ \hline \mathbf{S}_{\beta\alpha} & \mathbf{S}_{\beta\beta} \\ \hline \end{array}$$

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

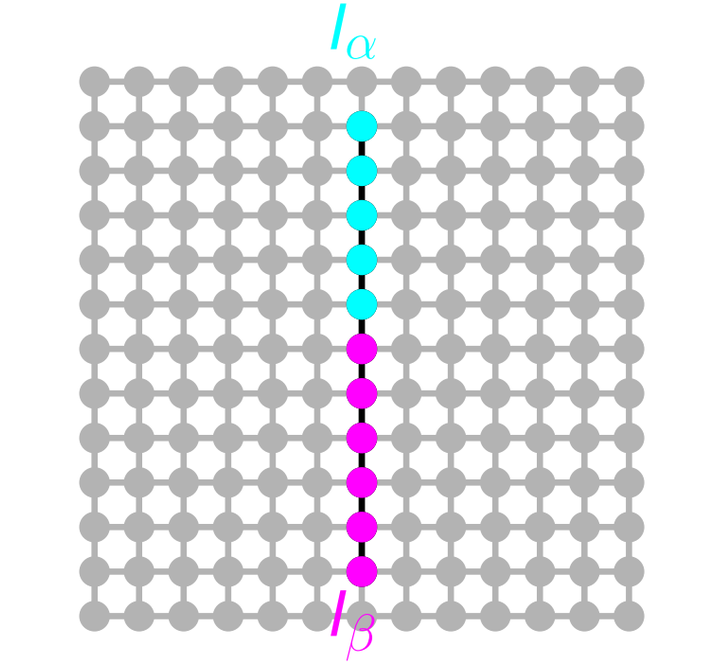
Recall our example of Laplace's equation discretized using the 5-point stencil.



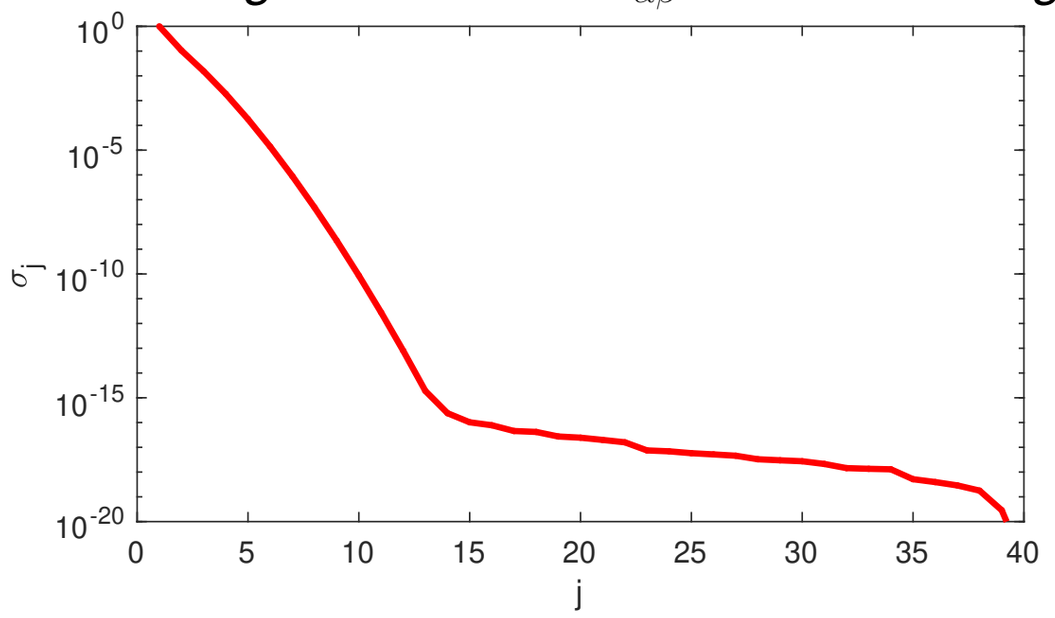
$$\mathbf{A} = \begin{array}{|c|c|c|} \hline \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \\ \hline \end{array}$$

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:



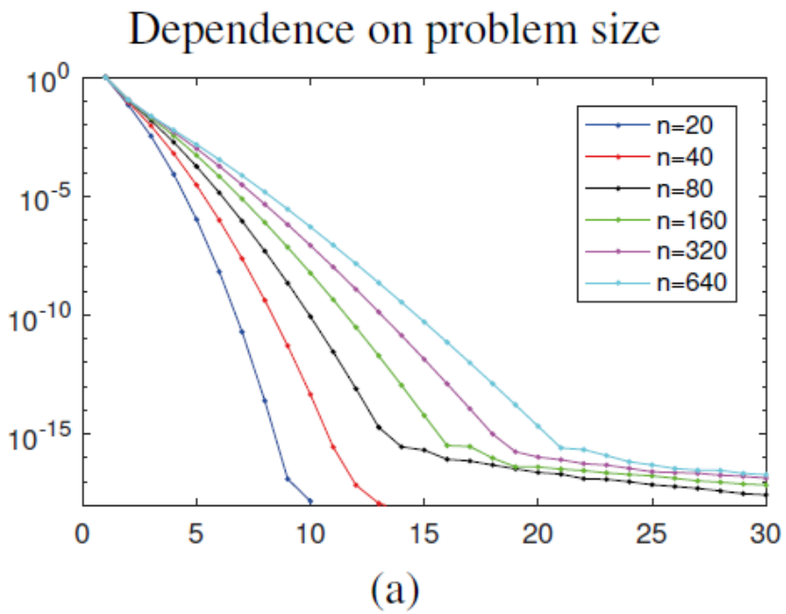
Singular values of $S_{\alpha\beta}$ for an 80×80 *grid.*

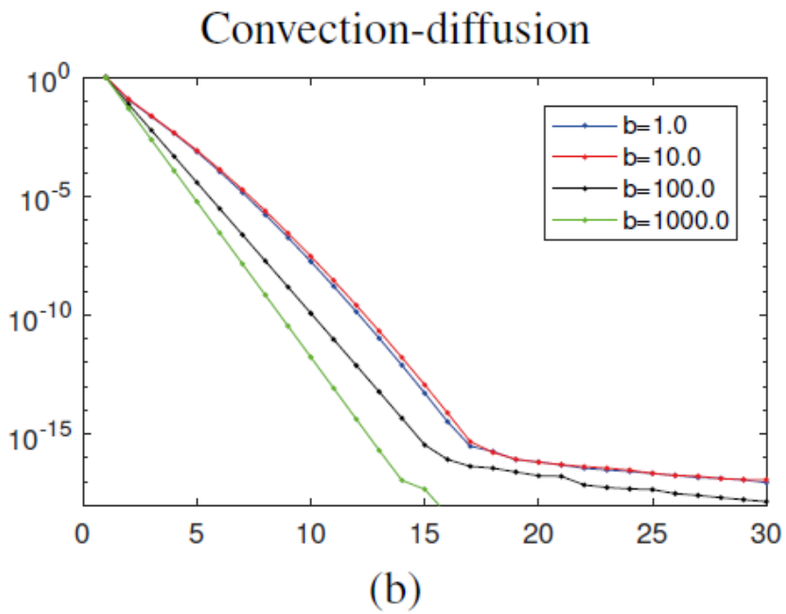


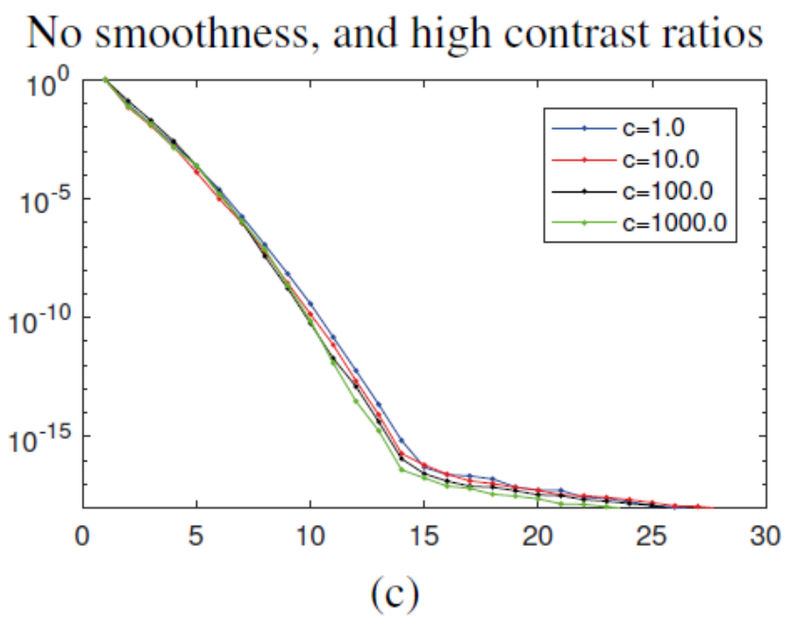
We explore the svds of $\mathbf{S}_{\alpha\beta}$ — encoding interactions between I_α and I_β.

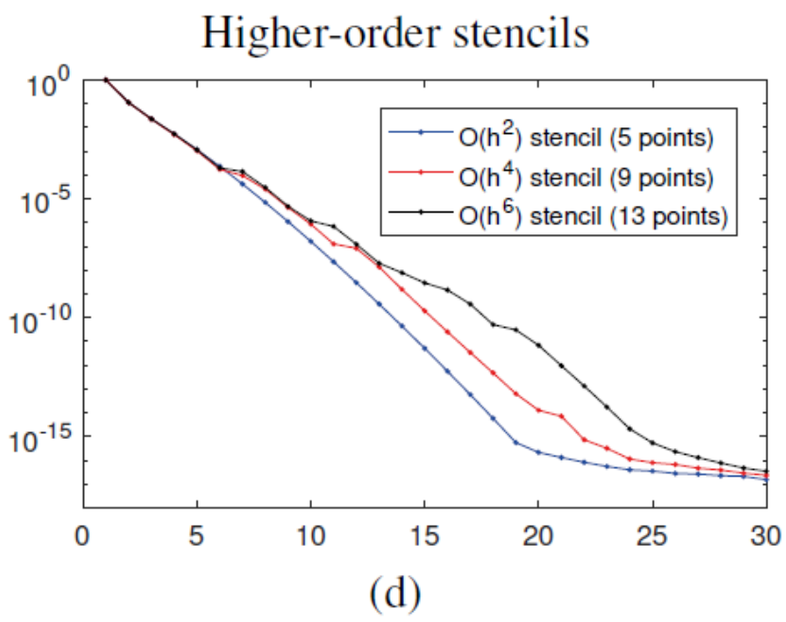
Interaction ranks: Stiffness matrix from finite difference discretization

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









Note: The rank decay property is remarkably stable!

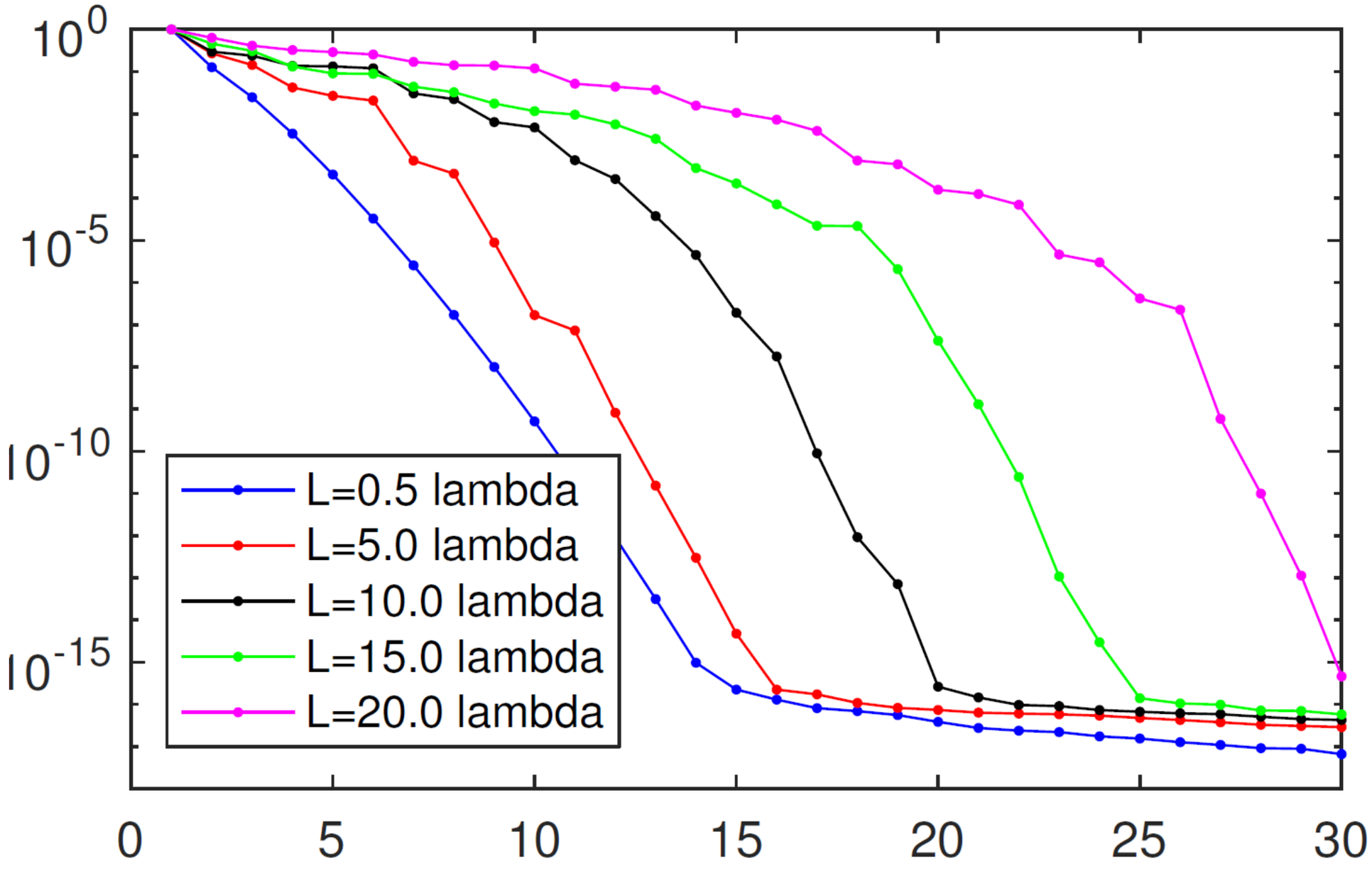
Note: The decay continues to ϵ_{mach} — regardless of the discretization errors!

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

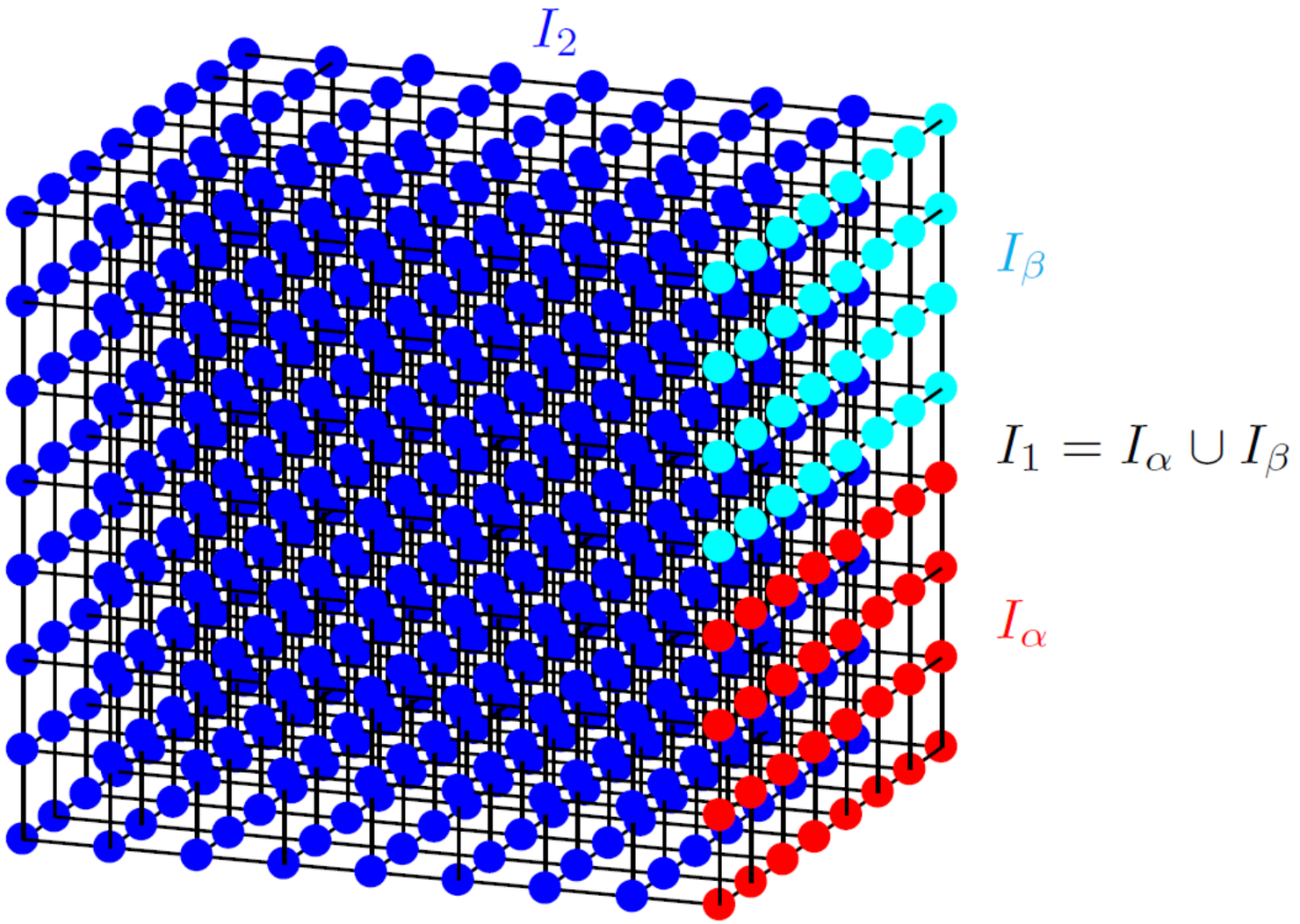
Next, let us consider Helmholtz problems with increasing wave numbers.



Fast decay *once oscillations are resolved.*

Interaction ranks: Stiffness matrix from finite difference discretization

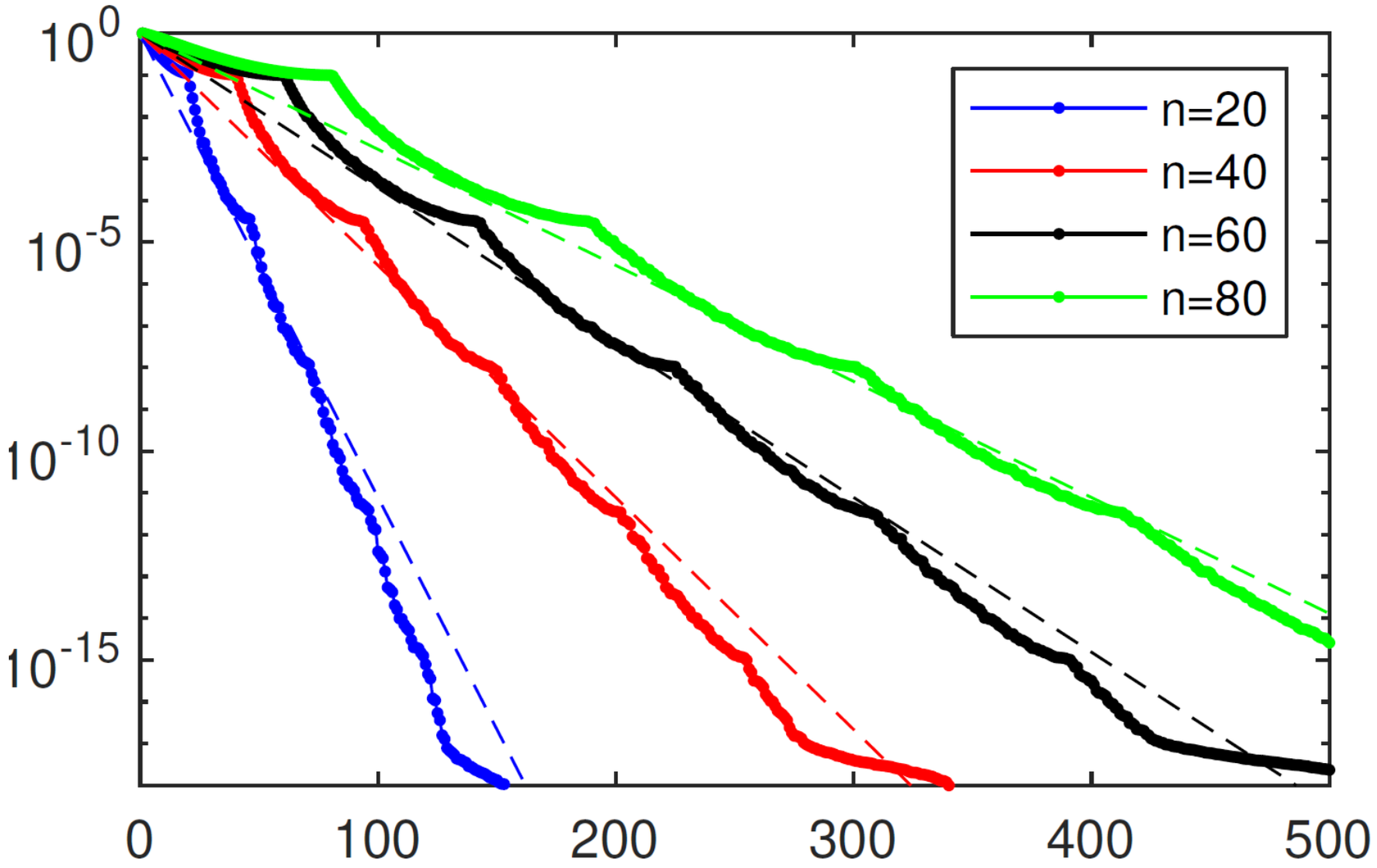
Finally, let us consider the analogous 3D problem.



The geometry.

Interaction ranks: Stiffness matrix from finite difference discretization

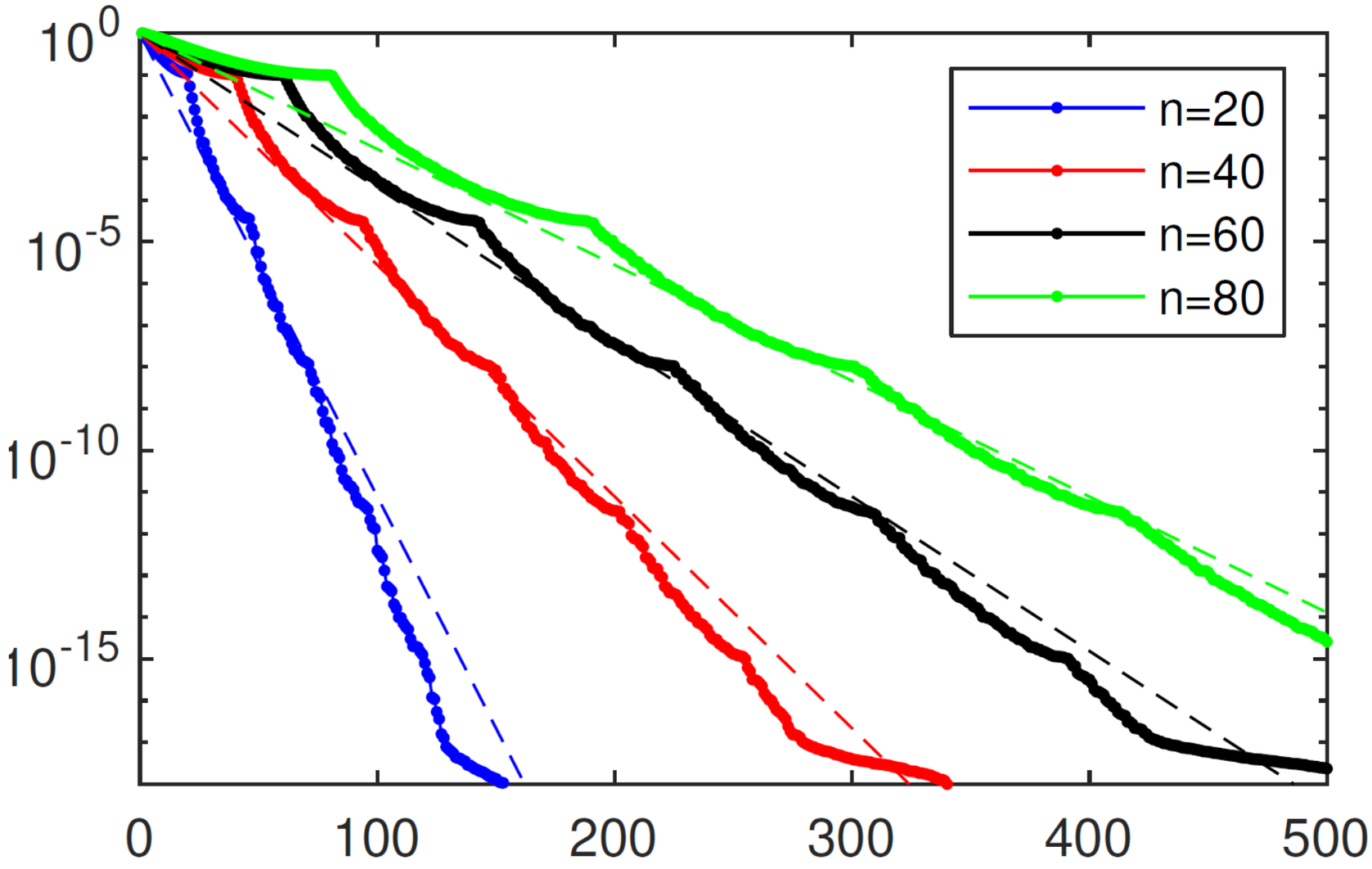
Finally, let us consider the analogous 3D problem.



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. *$\mathcal{H}$-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?

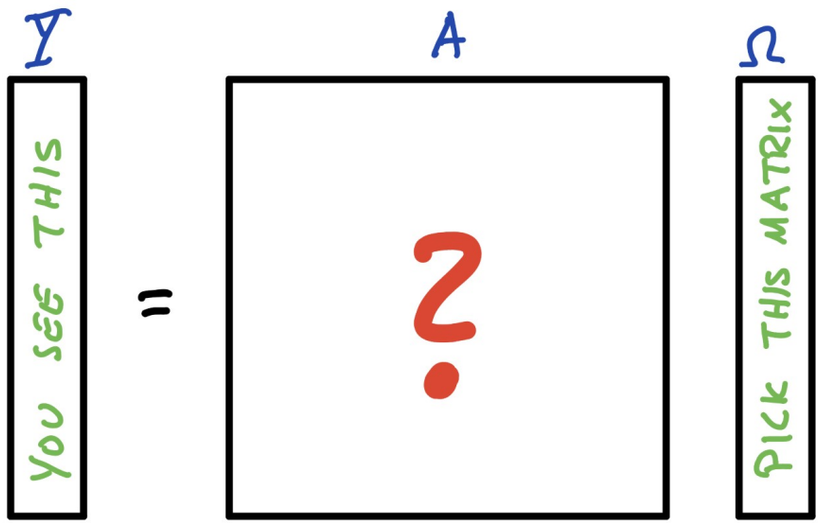
- In some standard environments (convolution with a known fundamental solution, say), there exist analytic techniques that work very well. "Abramowitz & Stegun" or "proxy surface method". *Classical FMM environment*
- In some cases where the kernel is known explicitly, heuristic techniques such as "adaptive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

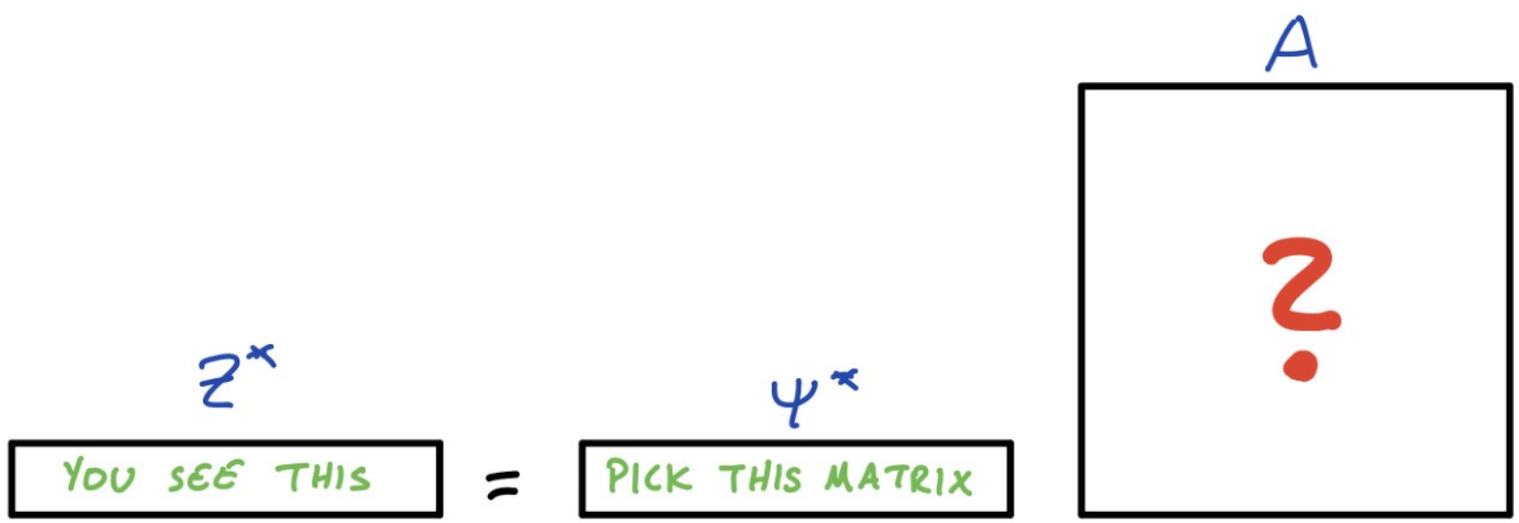
Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

Objective: Construct thin matrices $\mathbf{\Omega}$ 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 $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$?

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 ran($\mathbf{Y}$), e.g., $[\mathbf{\Psi}, \sim] = \mathtt{qr}(\mathbf{Y}, 0)$.
- Form $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$.

Then $\mathbf{A} = \mathbf{\Psi}\left(\mathbf{\Psi}^*\mathbf{A}\right) = \mathbf{\Psi}\mathbf{Z}^*$ 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.

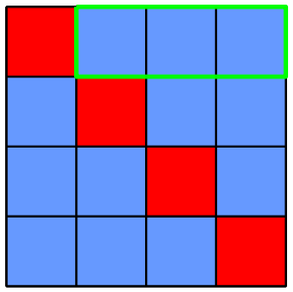
Even more recent: Randomized Simultaneous Factorization and Compression.

- A. Yesypenko, P.G. Martinsson, arXiv:2311.01451. To appear in J. of Sci. Comp.

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.

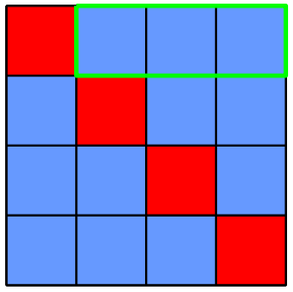
We seek a sample of $\mathbf{A}(I_4, I_4^c)$, the HBS row block of node 4.



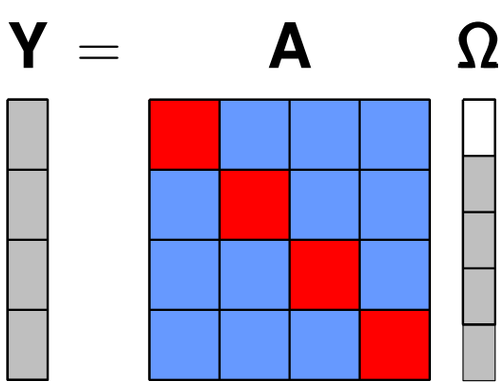
Compression of a rank-structured matrix – A naive approach

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.



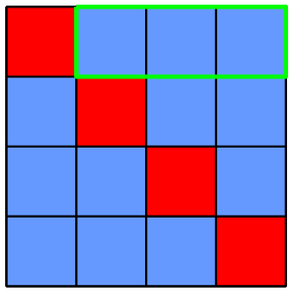
The naive approach is to sample with a random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4. Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^c)$.



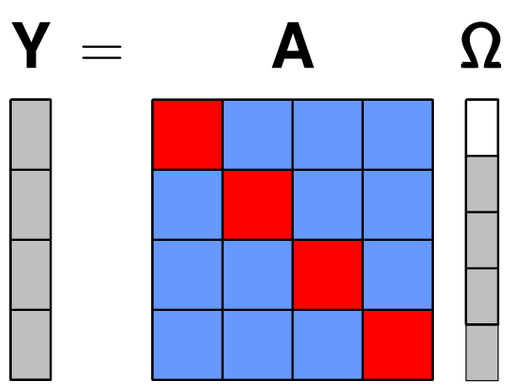
Compression of a rank-structured matrix – A naive approach

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 $\mathbf{Y}$.

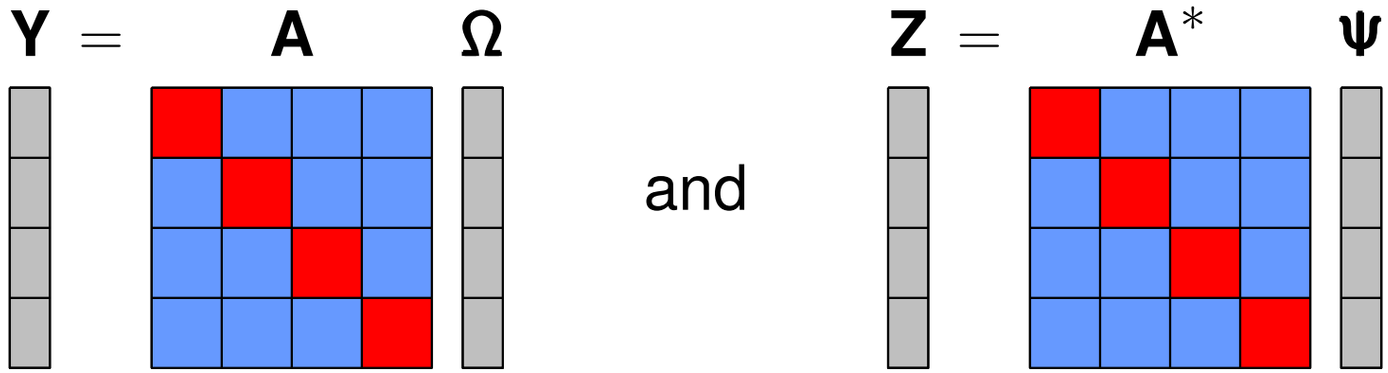
Compression of a rank-structured matrix – the "almost" black-box case

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

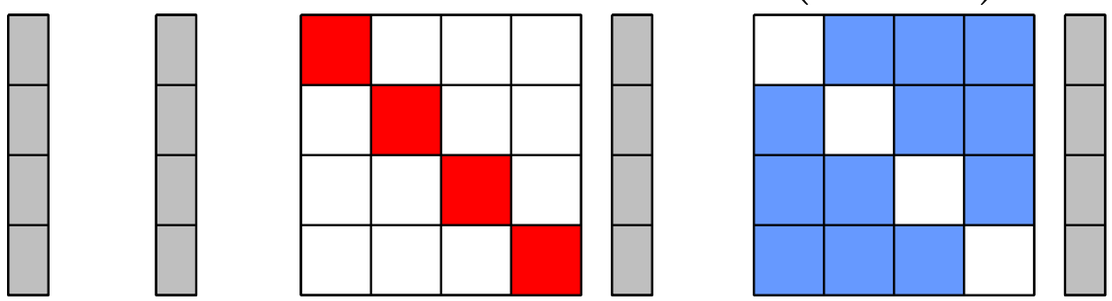
Sample $\mathbf{A}$ with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N\times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N\times r}$:



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

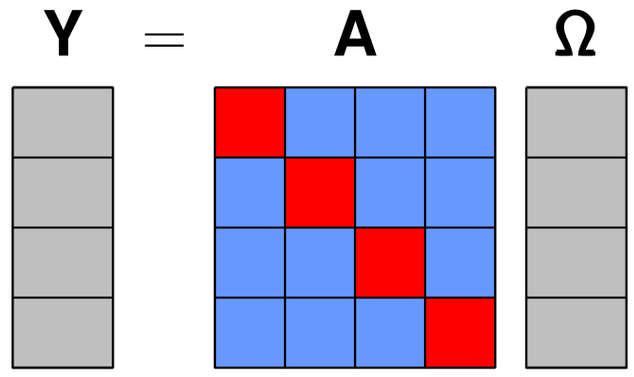
$$\mathbf{A}_{i,j} \approx \mathbf{Y}_i\,\mathbf{B}_{i,j}\,\mathbf{Z}_j^*, \qquad i \neq j,$$

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

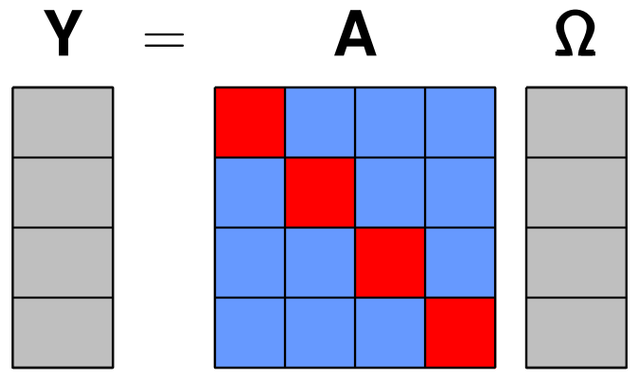
Sample $\mathbf{A}$ with a completely dense random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times (r+m)}$, where m is the leaf node size. (Think $m \approx 2k$ and $r = k + 10$.)



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



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

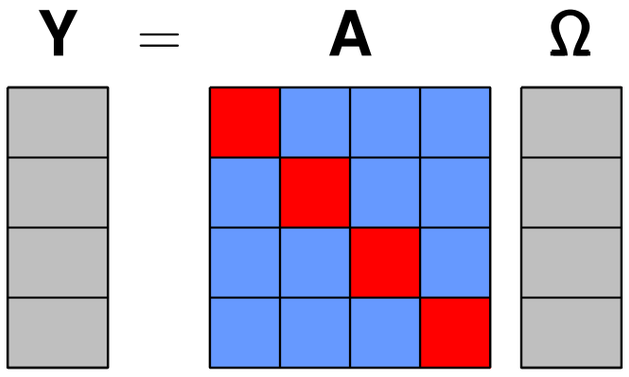
Since $\mathbf{\Omega}(I_4, :)$ is of size $m \times (r + m)$, it has a nullspace of dimension at least r. Let

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

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

Compression of a rank-structured matrix – fully black box

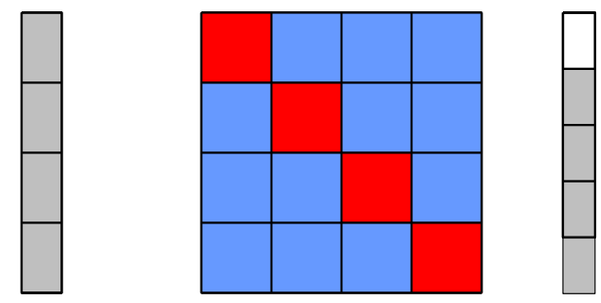
Sample **A** with a completely dense random matrix $\mathbf{\Omega} \in \mathbb{R}^{N\times(r+m)}$, where m is the leaf node size. (Think $m \approx 2k$ and $r = k + 10$.)



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

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

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

$\mathbf{YQ}_4 =$  $\mathbf{\Omega Q}_4$.

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

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

Compression of a rank-structured matrix – fully black box

- For each leaf node τ, we compute

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

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

- $\mathbf{U}_\tau$ only depends on $\mathbf{\Omega}(I_\tau, :)$ and $\mathbf{Y}(I_\tau, :)$.
- We only need $r + m$ samples to find $\mathbf{U}_\tau$ for every leaf node τ.
- $\mathbf{\Omega Q}_\tau$ 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* $\mathbf{A}^*$*.*

Form Gaussian random random matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ of size $N \times s$.

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

Compress level by level from finest to coarsest.

for level $\ell = L, L-1, \ldots, 0$ **do**

  for node τ in level ℓ **do**

    if τ is a leaf node **then**

$$\mathbf{\Omega}_\tau = \mathbf{\Omega}(I_\tau, :), \qquad \mathbf{\Psi}_\tau = \mathbf{\Psi}(I_\tau, :)$$

$$\mathbf{Y}_\tau = \mathbf{Y}(I_\tau, :), \qquad \mathbf{Z}_\tau = \mathbf{Z}(I_\tau, :)$$

    else

Let α and β denote the children of τ.

$$\mathbf{\Omega}_\tau = \begin{bmatrix} \mathbf{V}_\alpha^* \mathbf{\Omega}_\alpha \\ \mathbf{V}_\beta^* \mathbf{\Omega}_\beta \end{bmatrix}, \qquad \mathbf{\Psi}_\tau = \begin{bmatrix} \mathbf{U}_\alpha^* \mathbf{\Psi}_\alpha \\ \mathbf{U}_\beta^* \mathbf{\Psi}_\beta \end{bmatrix}$$

$$\mathbf{Y}_\tau = \begin{bmatrix} \mathbf{U}_\alpha^* (\mathbf{Y}_\alpha - \mathbf{D}_\alpha \mathbf{\Omega}_\alpha) \\ \mathbf{U}_\beta^* (\mathbf{Y}_\beta - \mathbf{D}_\beta \mathbf{\Omega}_\beta) \end{bmatrix}, \quad \mathbf{Z}_\tau = \begin{bmatrix} \mathbf{V}_\alpha^* (\mathbf{Z}_\alpha - \mathbf{D}_\alpha^* \mathbf{\Psi}_\alpha) \\ \mathbf{V}_\beta^* (\mathbf{Z}_\beta - \mathbf{D}_\beta^* \mathbf{\Psi}_\beta) \end{bmatrix}$$

    if level $\ell > 0$ **then**

$$\mathbf{Q}_\tau = \texttt{nullspace}(\mathbf{\Omega}_\tau, r), \qquad \mathbf{P}_\tau = \texttt{nullspace}(\mathbf{\Psi}_\tau, r)$$

$$\mathbf{U}_\tau = \texttt{qr}(\mathbf{Y}_\tau \mathbf{Q}_\tau, r), \qquad \mathbf{V}_\tau = \texttt{qr}(\mathbf{Z}_\tau \mathbf{P}_\tau, r)$$

$$\mathbf{D}_\tau = (\mathbf{I} - \mathbf{U}_\tau \mathbf{U}_\tau^*)\mathbf{Y}_\tau \mathbf{\Omega}_\tau^\dagger + \mathbf{U}_\tau \mathbf{U}_\tau^* \left((\mathbf{I} - \mathbf{V}_\tau \mathbf{V}_\tau^*)\mathbf{Z}_\tau \mathbf{\Psi}_\tau^\dagger\right)^*$$

    else

$$\mathbf{D}_\tau = \mathbf{Y}_\tau \mathbf{\Omega}_\tau^\dagger$$

Compression of a rank-structured matrix – Finding 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 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 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.

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

Block $\mathbf{D}_\tau$ 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 $\boldsymbol{\Omega}$ to obtain

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

and rearrange to obtain

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

Compression of a rank structured matrix: RSRS

Suppose you have extracted samples

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

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

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

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 $\rightarrow$ very high practical speed.
  - Particularly efficient for GPUs, out-of-core computing, distributed memory, etc.
  - Structured random maps ("fast J-L transform"): Improves on $O(mnk)$ complexity.
  - 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.

- Recent *optimal* low rank compression scheme: IterativeCUR
  - A posteriori error estimation; rank adaptivity; sketch recycling.
  - Complexity is $O(mn + k^2(m + n))$ for rank-k approximation of $m \times n$ matrix.

- 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 M. O'Neil, "Fast Direct Solvers for Elliptic PDEs".
  In review, 2025. (Arxiv report 2511.07773)
  Review of direct algorithms for global operators in scientific computing.
- 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.

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