Randomized Linear Algebra at Scale: From Sketches to Operators

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, Nathaniel Pritchard, Gregorio Quintana-Ortí, Taejun Park, Vladimir Rokhlin, Joel Tropp, Mark Tygert, Heather Wilber.

Current and recent students/postdocs: Paul Beckman, Yunhui Cai, Chao Chen, Simon Dirckx, Yijun Dong, Michael Han, Joseph Kump, James Levitt, Kate Pearce, Anna Yesypenko.

Research support by:

Subject of talk: Efficient algorithms for standard linear algebraic tasks such as:

- Low rank approximation.
- Solving linear systems.
- Eigenvalue problems.

Why look for new methods? What changed?

Subject of talk: Efficient algorithms for standard linear algebraic tasks such as:

- Low rank approximation.
- Solving linear systems.
- Eigenvalue problems.

Why look for new methods? What changed?

Textbook methods for linear algebraic computations were designed for an environment where the main concern was *arithmetic cost*.

On modern hardware, different costs often dominate:

- moving data between levels of memory,
- moving data across processors,
- making repeated passes over a massive matrix.

Randomized methods often require much less data movement.

(In some cases, they have much lower asymptotic flop counts too!)

Subject of talk: Efficient algorithms for standard linear algebraic tasks such as:

- Low rank approximation.
- Solving linear systems.
- Eigenvalue problems.

Basic algorithmic template:

- Take a matrix or operator whose size, or repeated appearance, makes a computation expensive.
- Probe it by observing its action on a small number of randomized test vectors.
  These probes, or "sketches", can often be extracted in parallel.
- Recover the parts that matter, and discard components that got caught accidentally.

$$\textit{Probe} \rightarrow \textit{Recover structure} \rightarrow \textit{Compute on the compressed object}$$

In many important cases, this yields:

- high accuracy,
- major speedups,
- much less communication,
- access to problems that were previously out of reach.

Outline

- Low-rank approximation – “randomized SVD (RSVD)”.

  Well-established material. Background.

- A posteriori error estimation and rank adaptivity.

  Certified accuracy, adaptive rank selection, and automated codes.

- Data-sparse representations of global operators in scientific computing.

  Extending the RSVD methodology to PDE solution operators, DtNs, evolution operators, . . .

  Not globally low rank. “Hierarchical matrices”.

  Objective: Linear complexity direct solvers for challenging PDEs.

- If time permits:

  “Block nullification” and “randomized simultaneous compression and factorization”.

  How to actually execute the compression step for global operators.

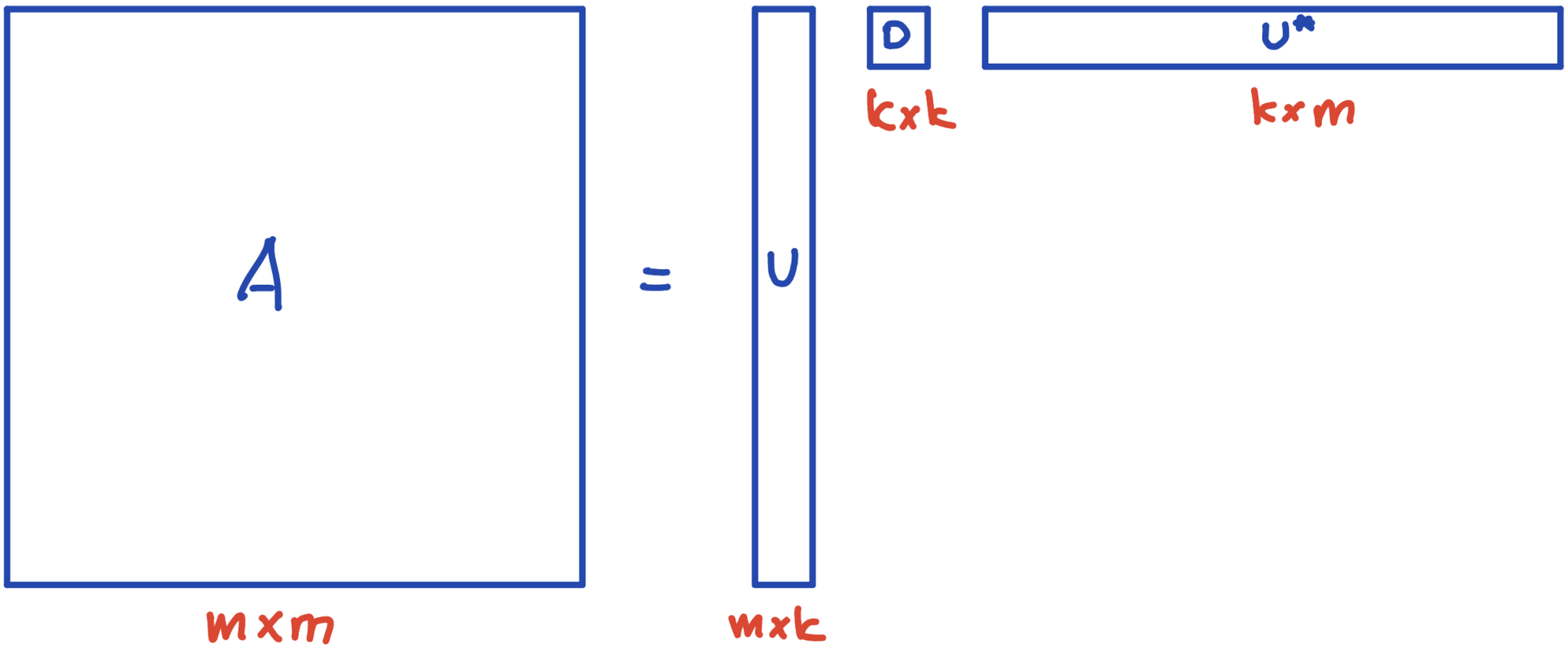
Low rank approximation:

Let **A** be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

For instance

$$\underset{m\times m}{\mathbf{A}} \approx \underset{m\times k}{\mathbf{U}}\ \underset{k\times k}{\mathbf{D}}\ \underset{k\times m}{\mathbf{U}^*}.$$



Retain the essential action of **A** using far fewer numbers.

Low rank approximation:

Let **A** be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Low rank approximation:

Let **A** be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Classical answers:

- Gram-Schmidt on the columns of **A**.
- Power iteration to find eigenvectors.
- Krylov methods.
- …

Low rank approximation:

Let **A** be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Randomized algorithm:

Warm-up case: Suppose that **A** has *exact* rank k and is positive semi-definite.

Draw an $m \times k$ Gaussian matrix $\mathbf{\Omega}$ and form the product $\mathbf{A\Omega}$. Then, with probability 1,

(NYS)
$$\underset{m \times m}{\mathbf{A}} = \underset{m \times k}{(\mathbf{A\Omega})} \ \underset{k \times k}{(\mathbf{\Omega}^*\mathbf{A\Omega})^{-1}} \ \underset{k \times m}{(\mathbf{A\Omega})^*}.$$

Low rank approximation:

Let $\mathbf{A}$ be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Randomized algorithm:

Warm-up case: Suppose that $\mathbf{A}$ has *exact* rank k and is positive semi-definite.

Draw an $m \times k$ Gaussian matrix $\mathbf{\Omega}$ and form the product $\mathbf{A\Omega}$. Then, with probability 1,

(NYS)
$$\underset{m \times m}{\mathbf{A}} = \underset{m \times k}{(\mathbf{A\Omega})} \; \underset{k \times k}{(\mathbf{\Omega}^*\mathbf{A\Omega})^{-1}} \; \underset{k \times m}{(\mathbf{A\Omega})^*}.$$

Proof: Let the exact eigenvalue decomposition of $\mathbf{A}$ be $\mathbf{A} = \mathbf{UDU}^*$. Then

$$\mathbf{A\Omega} = \mathbf{UDU}^*\mathbf{\Omega} = \{\mathbf{\Psi} := \mathbf{U}^*\mathbf{\Omega}\} = \mathbf{UD\Psi},$$

where $\mathbf{\Psi}$ is a $k \times k$ matrix that also has a Gaussian distribution. Then

$$(\mathbf{A\Omega})(\mathbf{\Omega}^*\mathbf{A\Omega})^{-1}(\mathbf{A\Omega})^* = (\mathbf{UD\Psi})(\mathbf{\Psi}^*\mathbf{D\Psi})^{-1}(\mathbf{UD\Psi})^*$$

Observe that $\mathbf{\Psi}$ is invertible with probability 1, so "almost surely"

$$(\mathbf{A\Omega})(\mathbf{\Omega}^*\mathbf{A\Omega})^{-1}(\mathbf{A\Omega})^* = \mathbf{UD\Psi}\,\mathbf{\Psi}^{-1}\mathbf{D}^{-1}\mathbf{\Psi}^{-*}\mathbf{\Psi}^*\mathbf{DU}^* = \mathbf{UDU}^* = \mathbf{A}.$$

Low rank approximation:

Let $\mathbf{A}$ be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Randomized algorithm:

Warm-up case: Suppose that $\mathbf{A}$ has *exact* rank k and is positive semi-definite.

Draw an $m \times k$ Gaussian matrix $\mathbf{\Omega}$ and form the product $\mathbf{A\Omega}$. Then, with probability 1,

$$\underset{m \times m}{\mathbf{A}} = \underset{m \times k}{(\mathbf{A\Omega})} \ \underset{k \times k}{(\mathbf{\Omega}^*\mathbf{A\Omega})^{-1}} \ \underset{k \times m}{(\mathbf{A\Omega})^*}. \tag{NYS}$$

Exploiting (NYS), you can easily compute an exact eval decomposition $\mathbf{A} = \mathbf{UDU}^*$:

1. Draw an $m \times k$ Gaussian random "test" matrix $\mathbf{\Omega}$. `Omega = randn(m,k)`
2. Form the $m \times k$ "sample" matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\mathrm{col}(\mathbf{Y}) = \mathrm{col}(\mathbf{Q})$. `[Q, ~] = qr(Y,0)`
4. Form the $k \times k$ symmetric matrix: $\mathbf{B} = \left(\mathbf{\Omega}^*\mathbf{Q}\right)^{-1}\mathbf{R}^*$. `B = inv(Omega'*Q)*R'`
5. Compute the EVD of $\mathbf{B}$ (small!): $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*$. `[Uhat, D] = eig(B)`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

The cost is $O(m^2k)$ for a dense matrix. *Same as classical methods!*

Low rank approximation:

Let **A** be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Randomized algorithm:

Warm-up case: Suppose that **A** has *exact* rank k and is positive semi-definite.

Draw an $m \times k$ Gaussian matrix $\mathbf{\Omega}$ and form the product $\mathbf{A\Omega}$. Then, with probability 1,

$$\underset{m \times m}{\mathbf{A}} = \underset{m \times k}{(\mathbf{A\Omega})} \ \underset{k \times k}{(\mathbf{\Omega}^*\mathbf{A\Omega})^{-1}} \ \underset{k \times m}{(\mathbf{A\Omega})^*}. \tag{NYS}$$

Exploiting (NYS), you can easily compute an exact eval decomposition $\mathbf{A} = \mathbf{UDU}^*$:

1. Draw an $m \times k$ Gaussian random "test" matrix $\mathbf{\Omega}$. `Omega = randn(m,k)`
2. Form the $m \times k$ "sample" matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\mathrm{col}(\mathbf{Y}) = \mathrm{col}(\mathbf{Q})$. `[Q, ~] = qr(Y,0)`
4. Forı `*Q)*R')`
5. Conı `eig(B)`
6. Forı `* Uhat`

> ☞ The interaction with **A** is only through the matrix-matrix product.
> Very high *wall clock* speed.
> Key reason for the practical impact of these methods.

The cost is $O(m^2k)$ for a dense matrix. *Same as classical methods!*

Low rank approximation:

Let **A** be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Randomized algorithm:

Warm-up case: Suppose that **A** has *exact* rank k and is positive semi-definite.

Draw an $m \times k$ Gaussian matrix $\mathbf{\Omega}$ and form the product $\mathbf{A\Omega}$. Then, with probability 1,

$$\underset{m \times m}{\mathbf{A}} = \underset{m \times k}{(\mathbf{A\Omega})} \ \underset{k \times k}{(\mathbf{\Omega}^*\mathbf{A\Omega})^{-1}} \ \underset{k \times m}{(\mathbf{A\Omega})^*}. \tag{NYS}$$

Observation:

The randomized algorithm described can be executed in a *streaming* mode.

You only need to see each entry of the matrix once!

The deterministic methods we described cannot do this.

Low rank approximation:

Let $\mathbf{A}$ be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Question: How do you compute the smaller factors efficiently?

Randomized algorithm:

Warm-up case: Suppose that $\mathbf{A}$ has *exact* rank k and is positive semi-definite.

Draw an $m \times k$ Gaussian matrix $\boldsymbol{\Omega}$ and form the product $\mathbf{A}\boldsymbol{\Omega}$. Then, with probability 1,

$$\text{(NYS)} \qquad \underset{m \times m}{\mathbf{A}} = \underset{m \times k}{(\mathbf{A}\boldsymbol{\Omega})} \ \underset{k \times k}{(\boldsymbol{\Omega}^*\mathbf{A}\boldsymbol{\Omega})^{-1}} \ \underset{k \times m}{(\mathbf{A}\boldsymbol{\Omega})^*}.$$

Observation: We can improve on the asymptotic complexity.

The key is to use a *structured* random matrix $\boldsymbol{\Omega}$ for which $\mathbf{A}\boldsymbol{\Omega}$ can be evaluated rapidly.

- Randomly scrambled discrete Fourier transforms — apply via FFT.
- Chains of Givens rotations between pairs of columns. "Kac's random walk."
- The matrix $\boldsymbol{\Omega}$ can even be *sparse*. Need ≥ 2 non-zeros per row.

Reduction in asymptotic complexity from $O(m^2 k)$ to $O(m^2 + mk^2)$.

Low rank approximation:

Let **A** be a large matrix of numerically low rank. Say symmetric.

We seek to compute a low rank factorization.

Questic

Randor

Warm-u e.

Draw ar ability 1,

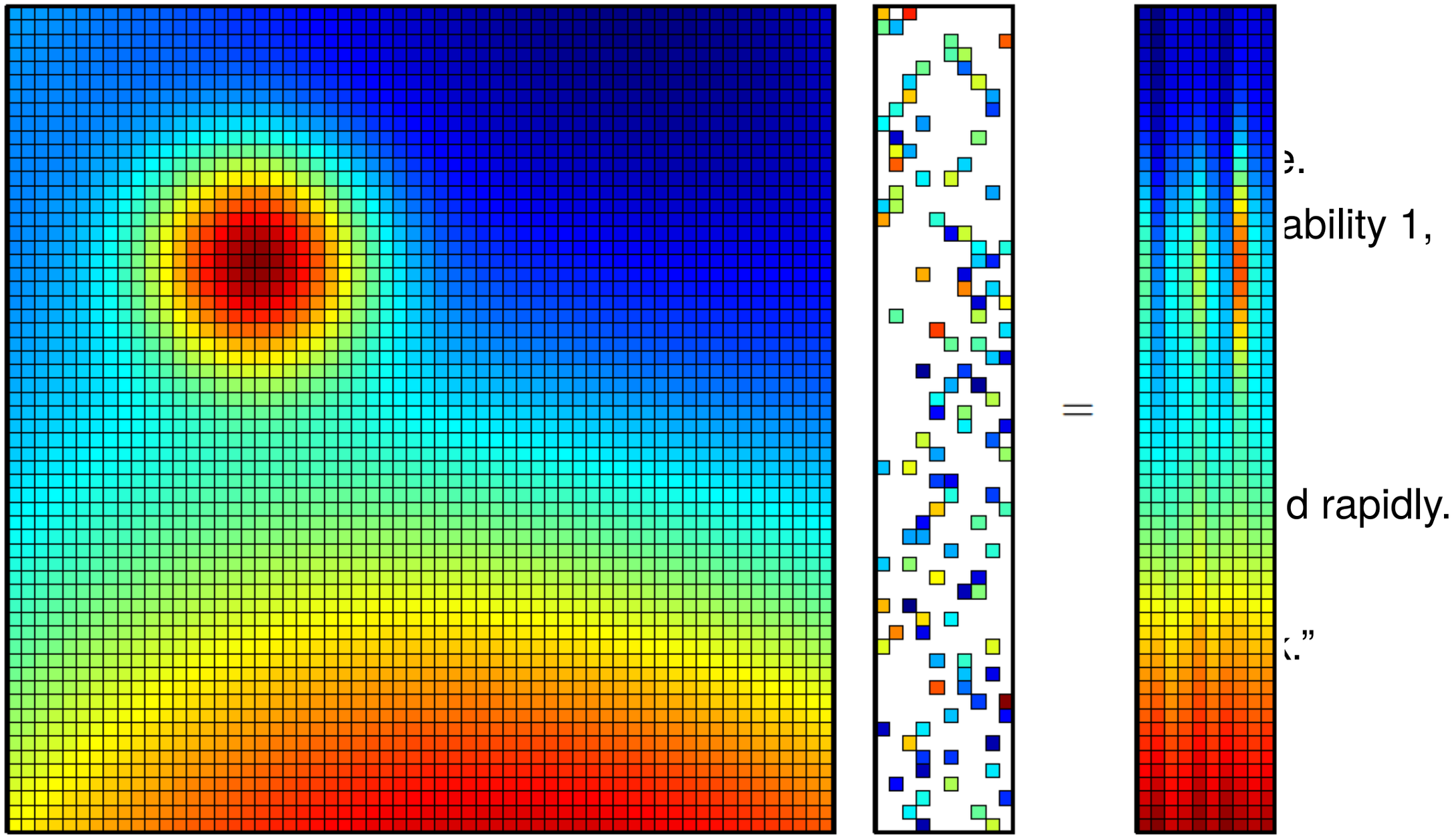
(NYS)

Observ

The key d rapidly.

- Rar
- Cha k."
- The

Reducti



Low rank approximation:

Next, let us widen the scope.

Warm-up case: **A** is $m \times m$ symmetric, non-negative, exact rank k, seek EVD:

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{D} & \mathbf{U}^*. \\ m \times m & & m \times k & k \times k & k \times m \end{array}$$

General case: **A** is $m \times n$, approximate rank k, we seek singular value decomposition:

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

Where **U** and **V** are orthonormal, and **D** is diagonal.

The key features of the algorithm remain:

- Draw a thin random matrix $\mathbf{\Omega}$.
- Build a sample matrix $\mathbf{A\Omega}$ whose columns approximately span ran($\mathbf{A}$).
- Interaction with **A** is only through matrix-matrix multiplication.
- Streaming mode ("single-view") is still possible.
- Structured $\mathbf{\Omega}$ leads to acceleration from $O(mnk)$ to $O(mn + (m+n)k^2)$.

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:

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{U}} \; \underset{k \times k}{\mathbf{D}} \; \underset{k \times n}{\mathbf{V}^*},$$

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

The "Randomized SVD (RSVD)":

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\text{col}(\mathbf{Y}) = \text{col}(\mathbf{Q})$. `[Q, ~] = qr(Y,0)`
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 $\mathbf{Y} = \mathbf{A\Omega}$ on line 2 by $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*\mathbf{\Omega})$, or $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*(\mathbf{A\Omega}))$, or ...

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:

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{U}} \; \underset{k \times k}{\mathbf{D}} \; \underset{k \times n}{\mathbf{V}^*},$$

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

The "Randomized SVD (RSVD)":

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\text{col}(\mathbf{Y}) = \text{col}(\mathbf{Q})$. `[Q, ~] = qr(Y,0)`
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`

Error analysis: Steps (3) – (6) are exact, so

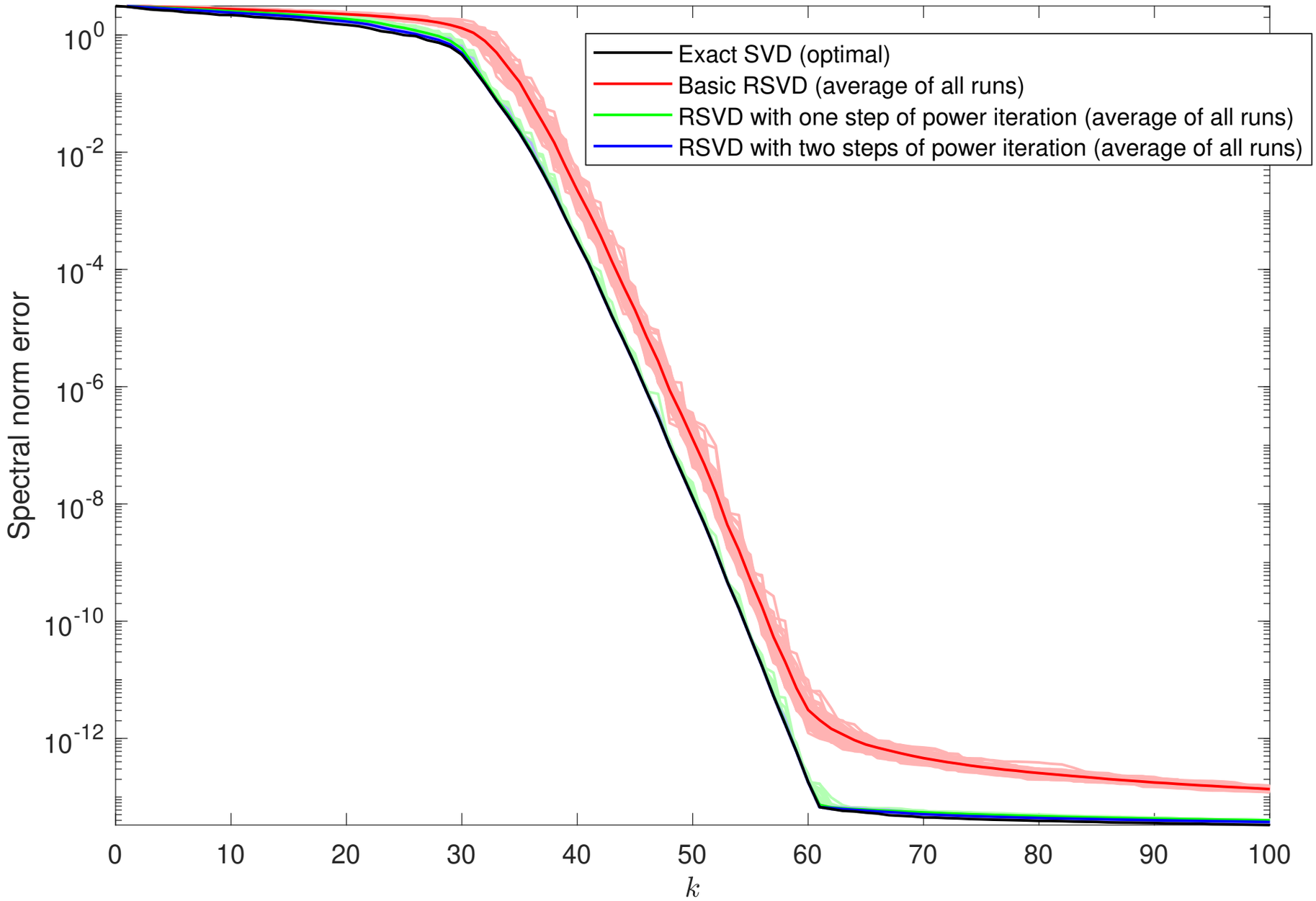
$$\mathbf{A}_{\text{approx}} := \mathbf{U}\mathbf{D}\mathbf{V}^* = \mathbf{Q}\hat{\mathbf{U}}\mathbf{D}\mathbf{V}^* = \mathbf{Q}\mathbf{B} = \mathbf{Q}\mathbf{Q}^*\mathbf{A} = \mathbf{P}_{\mathbf{Y}}\mathbf{A},$$

where $\mathbf{P}_{\mathbf{Y}}$ is the orthogonal projection onto $\text{span}(\mathbf{Y})$.

So the question is: How well do the columns of $\mathbf{Y} = \mathbf{A\Omega}$ span the columns of $\mathbf{A}$?

Low rank approximation:

The graphs show the error $e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\| = \|\mathbf{A} - \mathbf{P}_\mathbf{Y}\mathbf{A}\|$,
where $\mathbf{P}_\mathbf{Y}$ is the orthogonal projection onto $\text{col}(\mathbf{Y})$ with $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$.

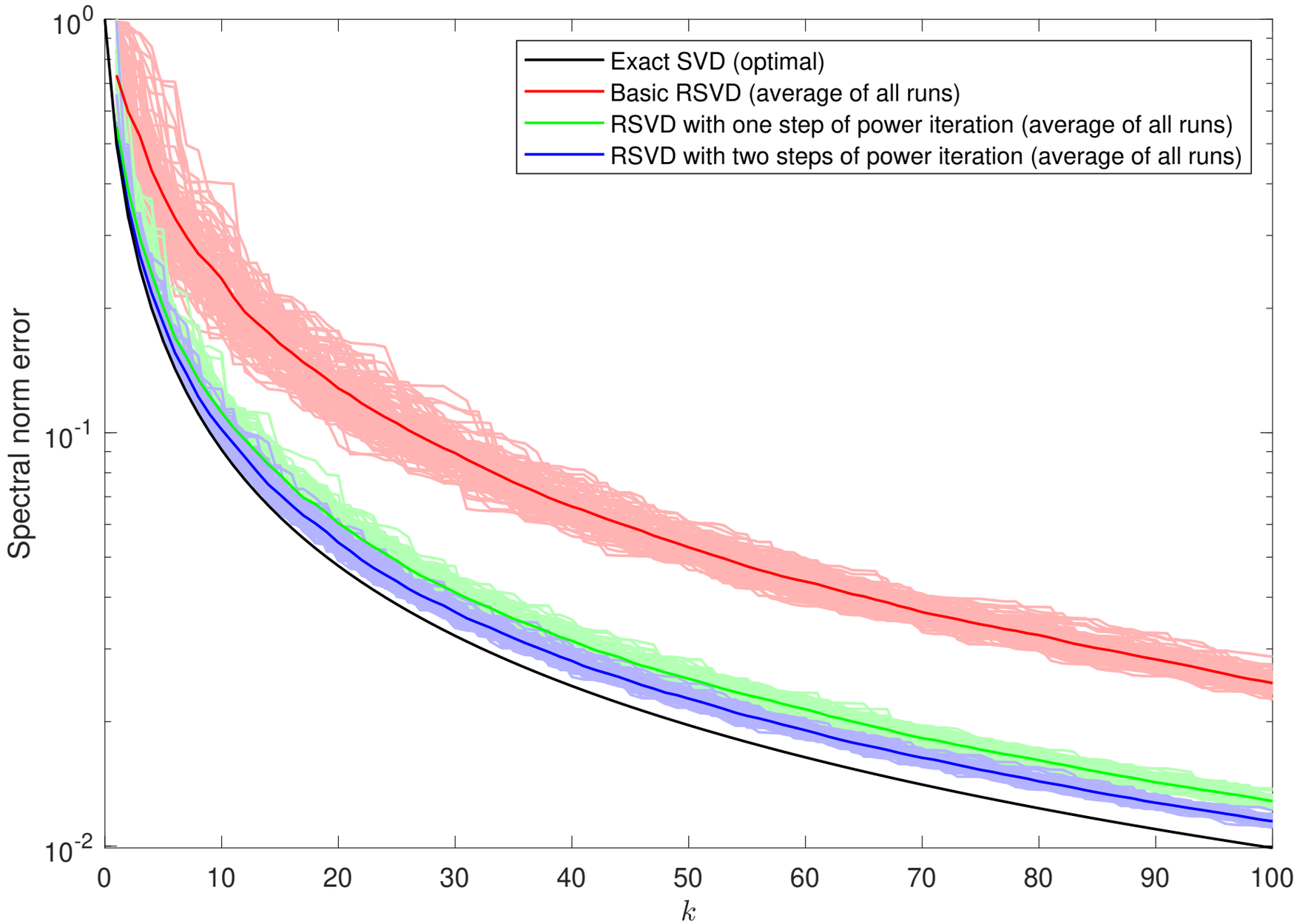


100 instantiations are shown. The darker lines are the empirical average error.

The example shown involves a discretized scattering operator. Fast decay.

Low rank approximation:

The graphs show the error $e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\| = \|\mathbf{A} - \mathbf{P}_\mathbf{Y}\mathbf{A}\|$,
where $\mathbf{P}_\mathbf{Y}$ is the orthogonal projection onto $\text{col}(\mathbf{Y})$ with $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$.

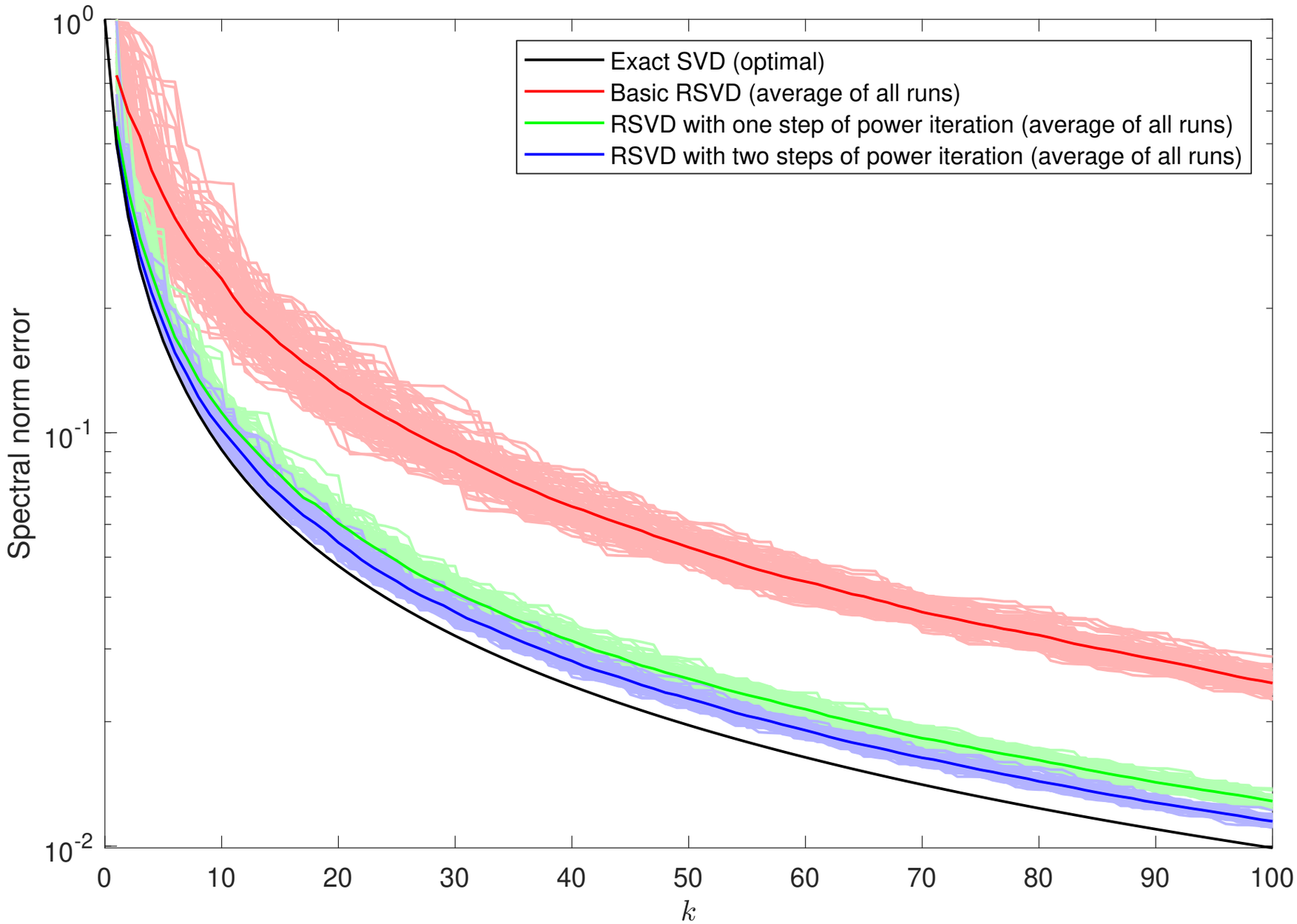


100 instantiations are shown. The darker lines are the empirical average error.

Slow decay of singular values. Typical of "data science" applications.

Low rank approximation:

The graphs show the error $e = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\| = \|\mathbf{A} - \mathbf{P}_\mathbf{Y}\mathbf{A}\|$,
where $\mathbf{P}_\mathbf{Y}$ is the orthogonal projection onto $\text{col}(\mathbf{Y})$ with $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$.



100 instantiations are shown. The darker lines are the empirical average error.

Important: Some *over-sampling* is essential. Draw a few extra samples!

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 $\boldsymbol{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$.	(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 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

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

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

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

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

There are also bounds on the likelihood of a large deviation from the expectation.

Draws on bounds from Chen & Dongarra (2005) on $\|\mathbf{G}^\dagger\|$ for Gaussian $\mathbf{G}$ of size $k \times k+p$, with p small.

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 $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(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.
- Power iteration. Replace $\mathbf{Y} = \mathbf{A\Omega}$ by $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*\mathbf{\Omega})$, or $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*(\mathbf{A\Omega}))$, or ...
- 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 Givens rotations ("Kac's random walk"). Cost is $O(mn\log(k))$.
  - Sparse random matrices: As few as 2 non-zeros per row. Cost is $O(mn)$!
- 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!*

Outline

- Randomized low-rank approximation – “randomized SVD (RSVD)”.

  Well-established material. Background.

- A posteriori error estimation and rank adaptivity.

  Certified accuracy, adaptive rank selection, and automated codes.

- Data-sparse representations of global operators in scientific computing.

  Extending the RSVD methodology to PDE solution operators, DtNs, evolution operators, . . .

  Not globally low rank. “Hierarchical matrices”.

  Objective: Linear complexity direct solvers for challenging PDEs.

- If time permits:

  “Block nullification” and “randomized simultaneous compression and factorization”.

  How to actually execute the compression step for global operators.

Recent work: 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?

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

Recent work: 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 $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$, and 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 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 $\mathbf{C} = \mathbf{A}\mathbf{G}$. Then use that for any matrix $\mathbf{H}$, we have

$$\mathbb{E}\left[\|\mathbf{H}\mathbf{G}\|_{\text{F}}^2\right] = q\|\mathbf{H}\|_{\text{F}}^2.$$

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

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

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: A posteriori error estimation

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

Recall from previous slide: Draw 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*, 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: A posteriori error estimation

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

Recall from previous slide: Draw random matrix $\mathbf{\Omega}$, set $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$, 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*, 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 $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{A}\mathbf{\Omega}) = \text{ran}(\mathbf{Y})$. 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 $\mathbf{\Psi} \in \mathbb{R}^{m \times s}$, with $s = O(k)$, then

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

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

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

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

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

Recent work: A posteriori error estimation

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

Recall from previous slide: Draw 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*, draw $\mathbf{G}$, set $\mathbf{C} = \mathbf{AG}$, and use $e^2 \approx \frac{1}{q}\|(\mathbf{I} - \mathbf{QQ}^*)\mathbf{C}\|_{\text{F}}^2$.

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

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

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

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

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

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

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

With Yuji Nakatsukasa. Forthcoming.

Recent work: 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: 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: 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: 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.

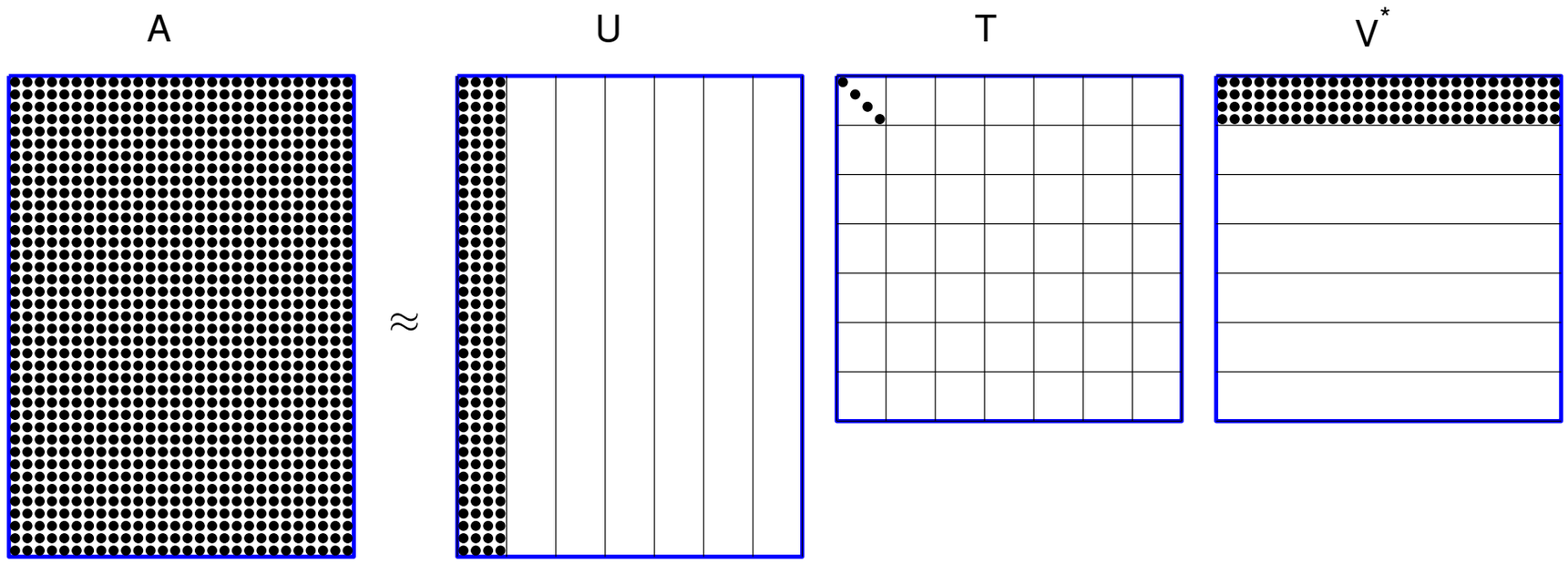
Recent work: Adaptive rank determination

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.

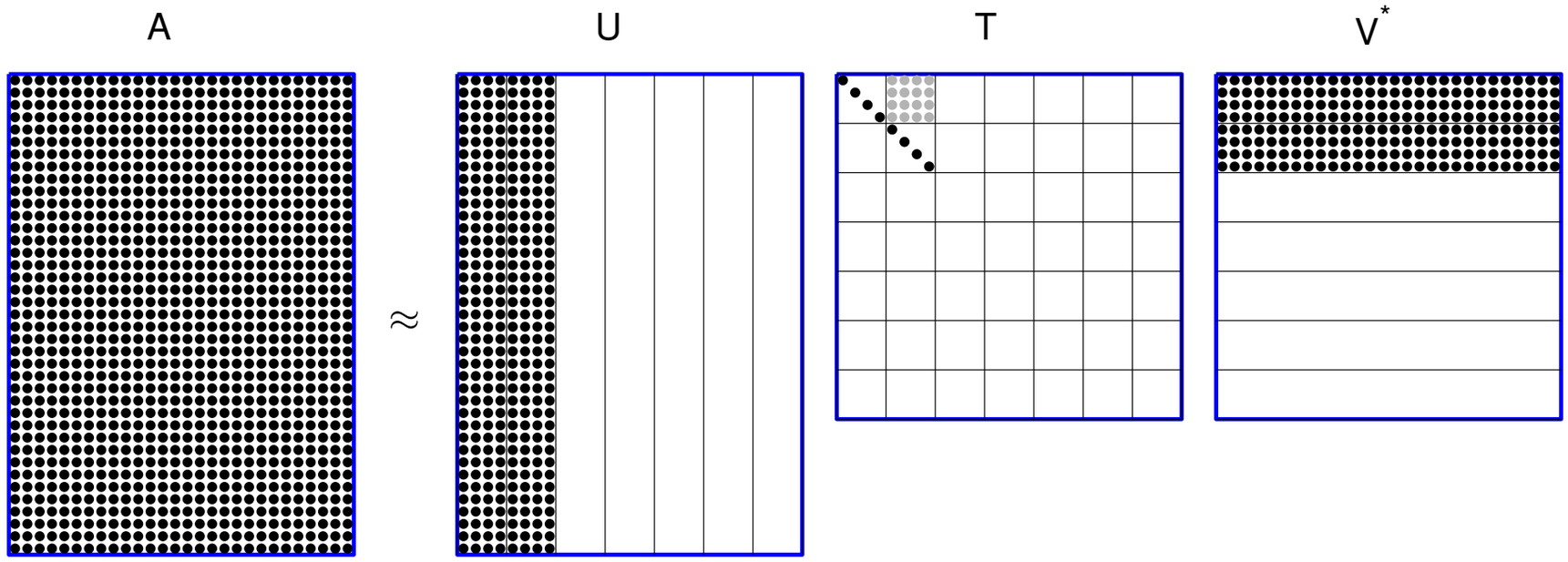
Recent work: Adaptive rank determination

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.

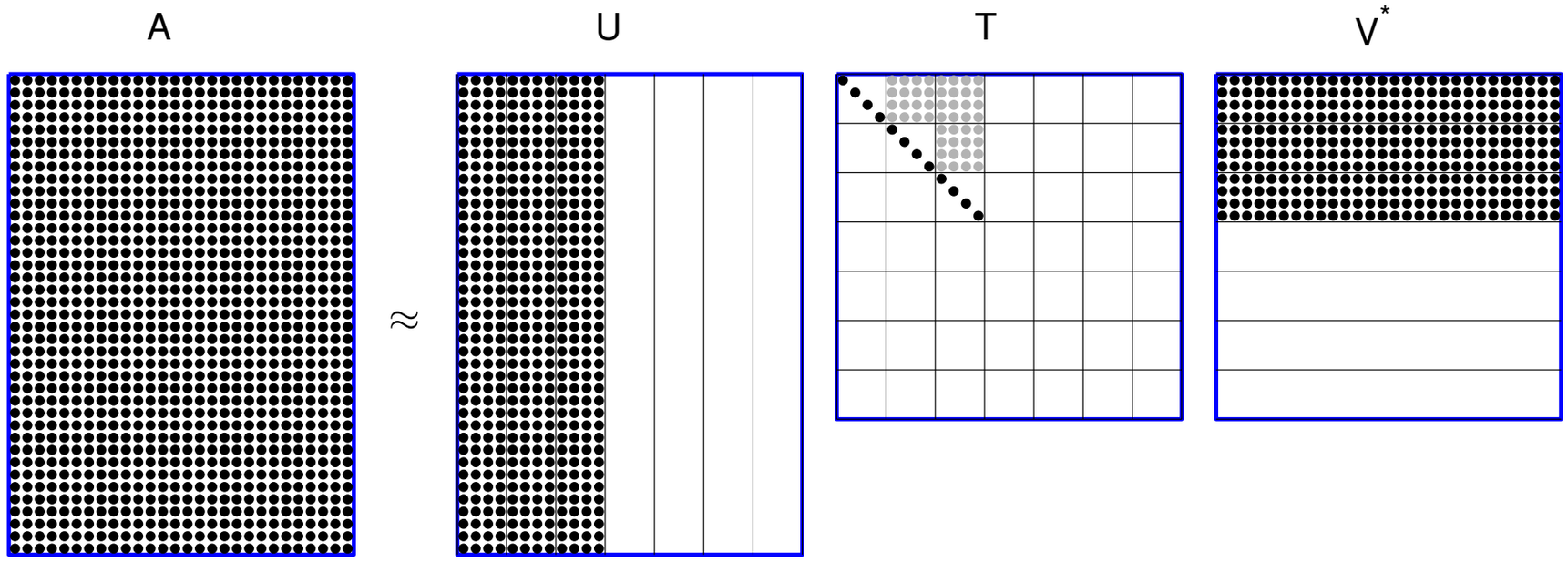
Recent work: Adaptive rank determination

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.

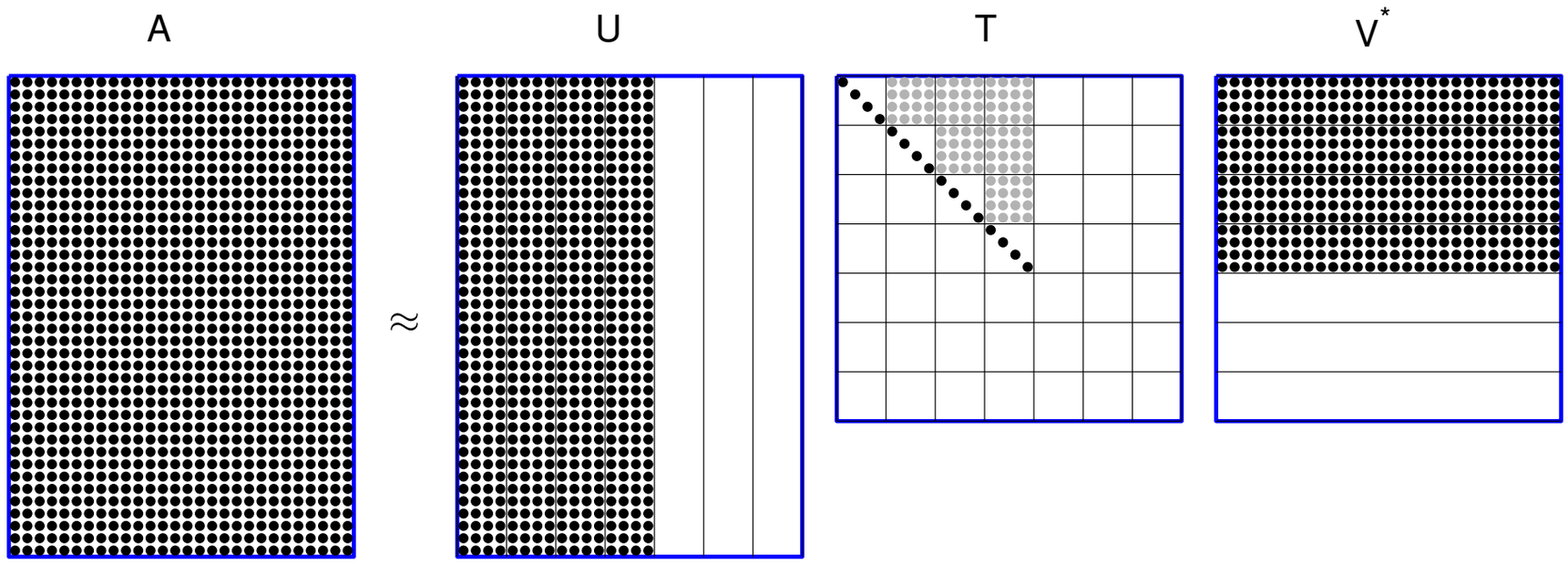
Recent work: Adaptive rank determination

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.

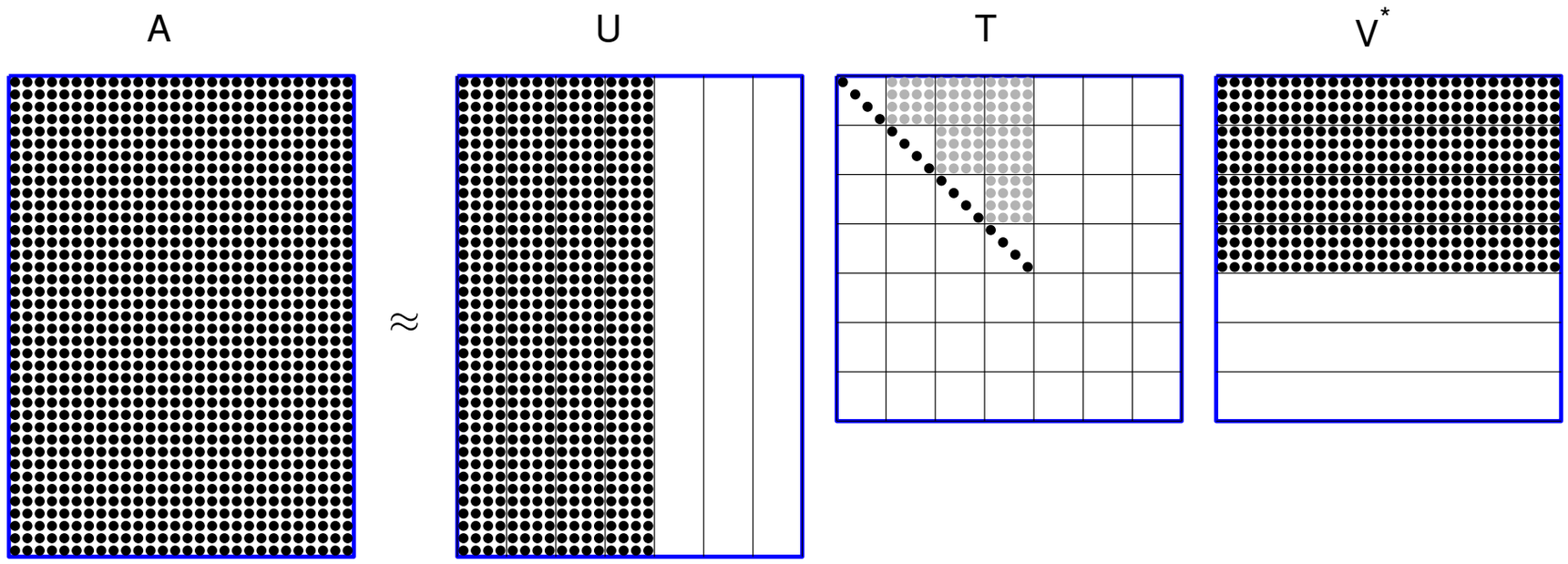
Recent work: Adaptive rank determination

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 $b = 4$:



Note: This process can be continued all the way to build a full *complete orthogonal decomposition*. This results in an $O(N^3)$ algorithm with very high practical speed. Particularly effective for data stored out-of-core. Matrices up to size $1M \times 1M$ can be handled on a regular workstation. *[N. Heavner, P.G. Martinsson, G. Quintana-Orti, CCPE, **35**(22), 2023.]* Similar techniques can be used to execute a full Householder QR with column pivoting using BLAS3 operations. *[P.G. Martinsson, G. Quintana-Orti, N. Heavner & R. van de Geijn, SISC, **39**(2), 2017. J.A. Duersch & M. Gu, SISC, **39**(4), 2017.]*

Recent work: 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: 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 $\mathbf{A}$:

$$\mathbf{Y}_1 = \mathbf{A}\boldsymbol{\Omega}_1.$$

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 $\boldsymbol{\Omega}_2$ from a Gaussian distribution and setting $\mathbf{Y}_2 = \mathbf{A}^{(1)}\boldsymbol{\Omega}_2$, we proceed as follows:

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

where $\boldsymbol{\Omega}_2 := \mathbf{V}_1^*\boldsymbol{\Omega}_1$.

Question: Does $\boldsymbol{\Omega}_2$ have a Gaussian distribution?

Recent work: 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 $\mathbf{A}$:

$$\mathbf{Y}_1 = \mathbf{A}\boldsymbol{\Omega}_1.$$

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 $\boldsymbol{\Omega}_2$ from a Gaussian distribution and setting $\mathbf{Y}_2 = \mathbf{A}^{(1)}\boldsymbol{\Omega}_2$, we proceed as follows:

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

where $\boldsymbol{\Omega}_2 := \mathbf{V}_1^*\boldsymbol{\Omega}_1$.

Question: Does $\boldsymbol{\Omega}_2$ have a Gaussian distribution?

No. But in practice there is no discernible difference.

Recent work: IterativeCUR

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

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

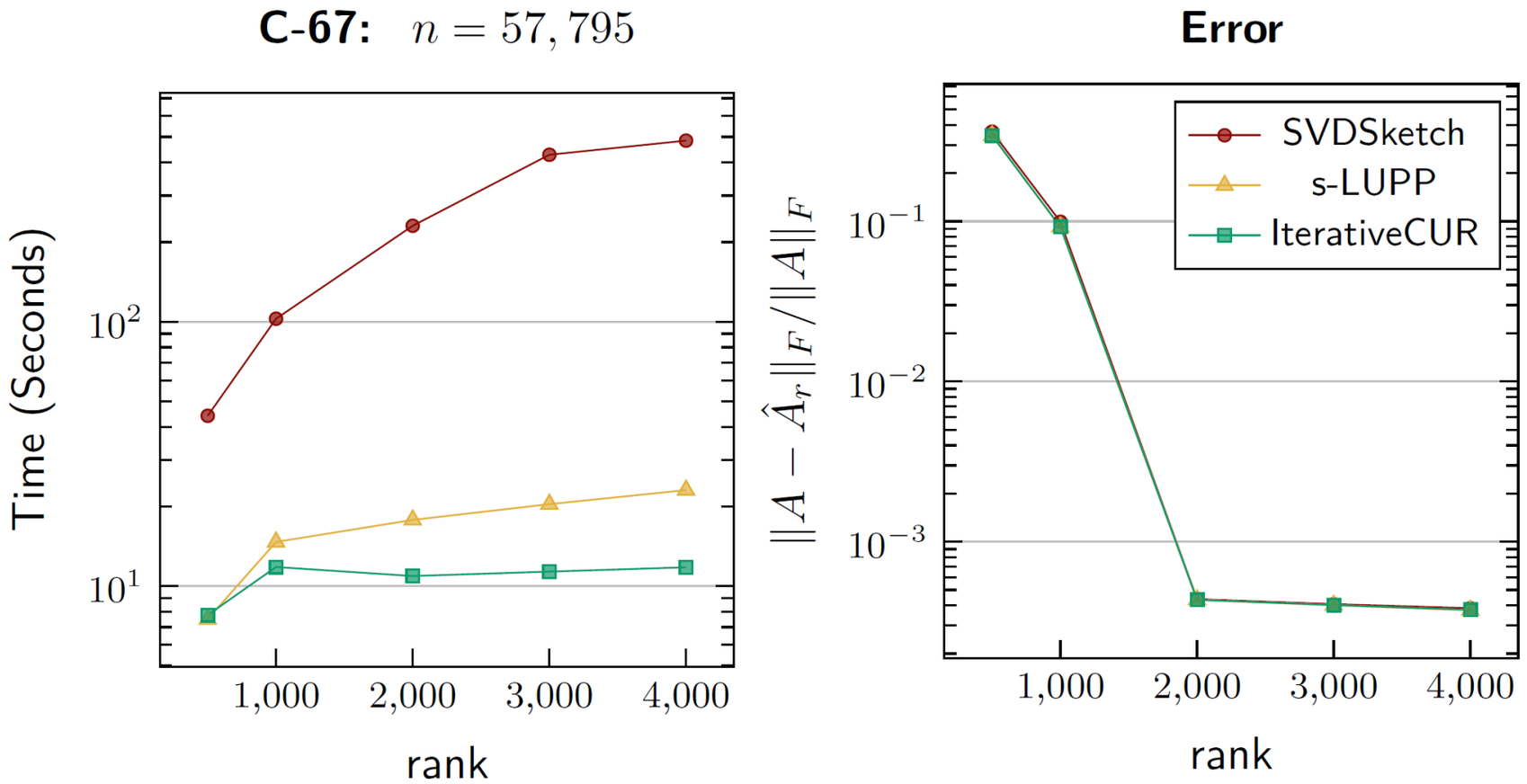
such that $\|\mathbf{A} - \mathbf{CUR}\|_{\mathrm{F}} \leq \varepsilon$.

Obtained by putting all the techniques described together:

- The method is *incremental and blocked* for computational efficiency.
- After each step of the iteration is completed, an *a posteriori error estimator* is used to provide a highly reliable stopping criterion.
- Relies on *interpolatory decompositions* that extract actual rows and columns of the matrix – this avoids the need for explicit updates!
- 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.

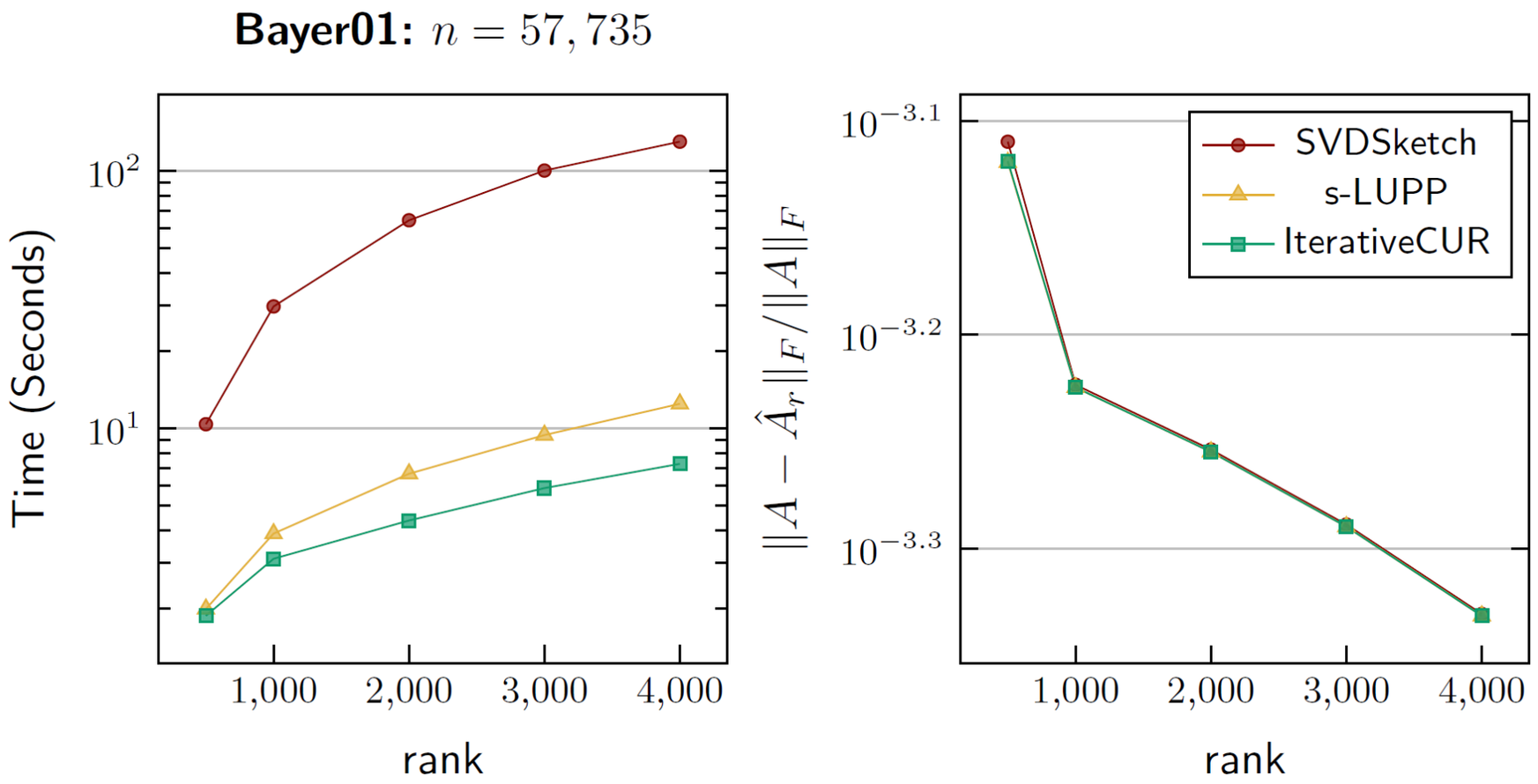
With N. Pritchard, T. Park, Y. Nakatsukasa, arxiv:2509.21963

Recent work: IterativeCUR – numerical examples



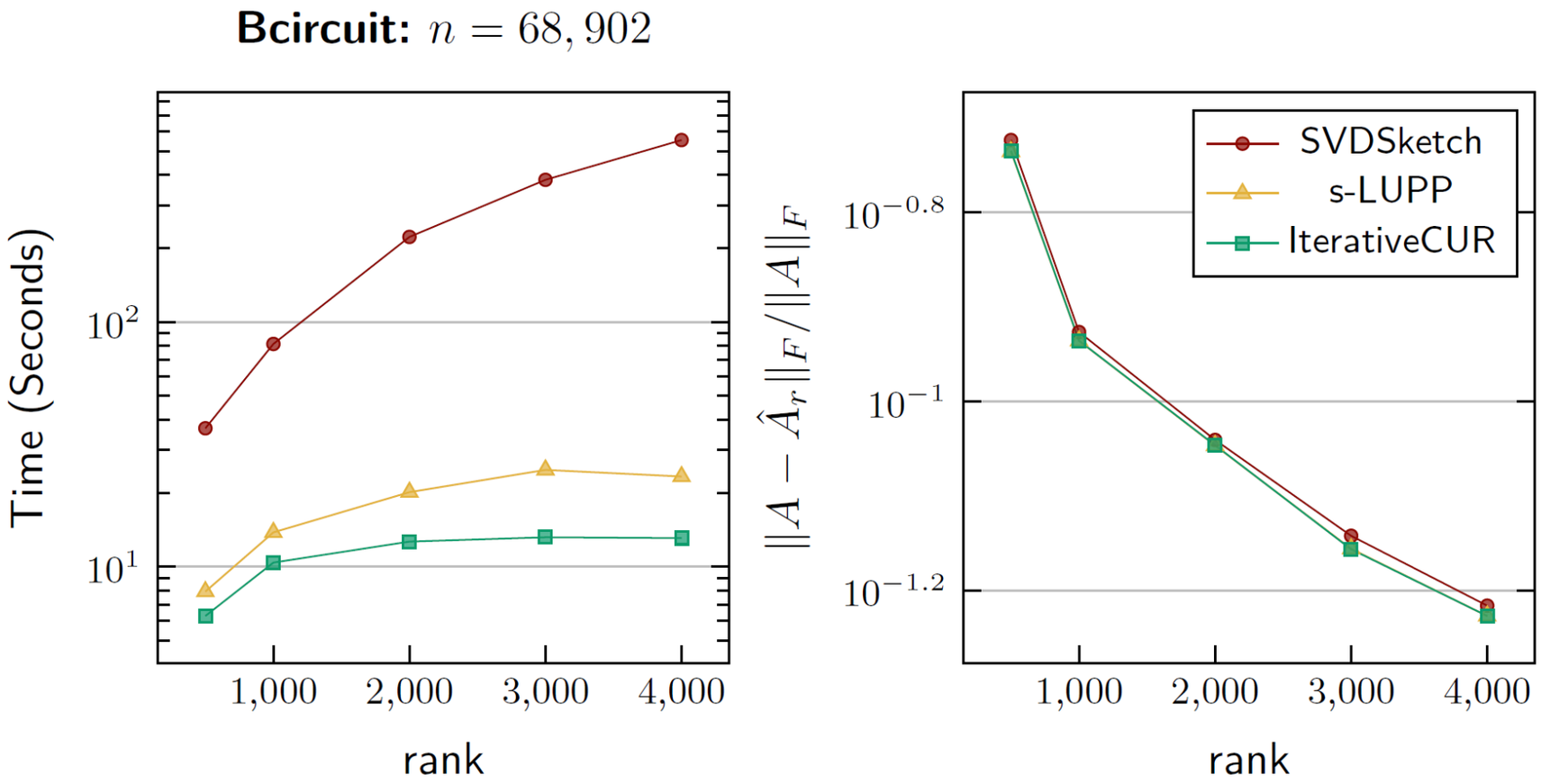
- SVDsketch: MATLAB's built-in [Yu-Yu-Li 18]
  - Roughly 'iterative RSVD' drawing new sketch each time
  - Cannot handle accuracy $\leq \epsilon_{\text{mach}}^{1/2} \approx 10^{-8}$
- s-LUPP: big sketch+LUPP
  - to test effect of reusing small sketch *G*
- IterativeCUR-LUPP: this work

Recent work: IterativeCUR – numerical examples



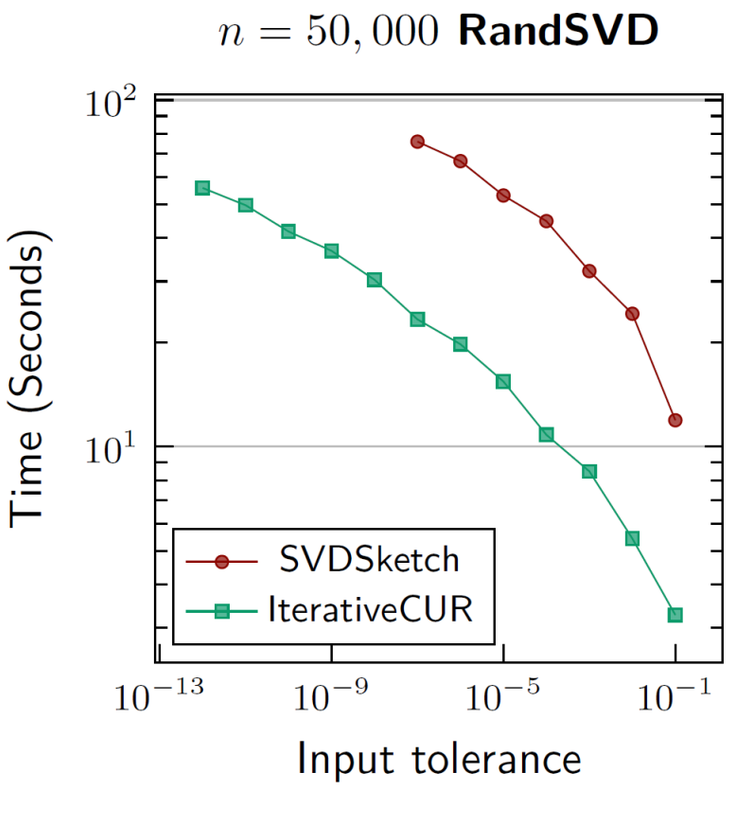
- SVDsketch: MATLAB's built-in [Yu-Yu-Li 18]
  - Roughly 'iterative RSVD' drawing new sketch each time
  - Cannot handle accuracy $\leq \epsilon_{\text{mach}}^{1/2} \approx 10^{-8}$
- s-LUPP: big sketch+LUPP
  - to test effect of reusing small sketch *G*
- IterativeCUR-LUPP: this work

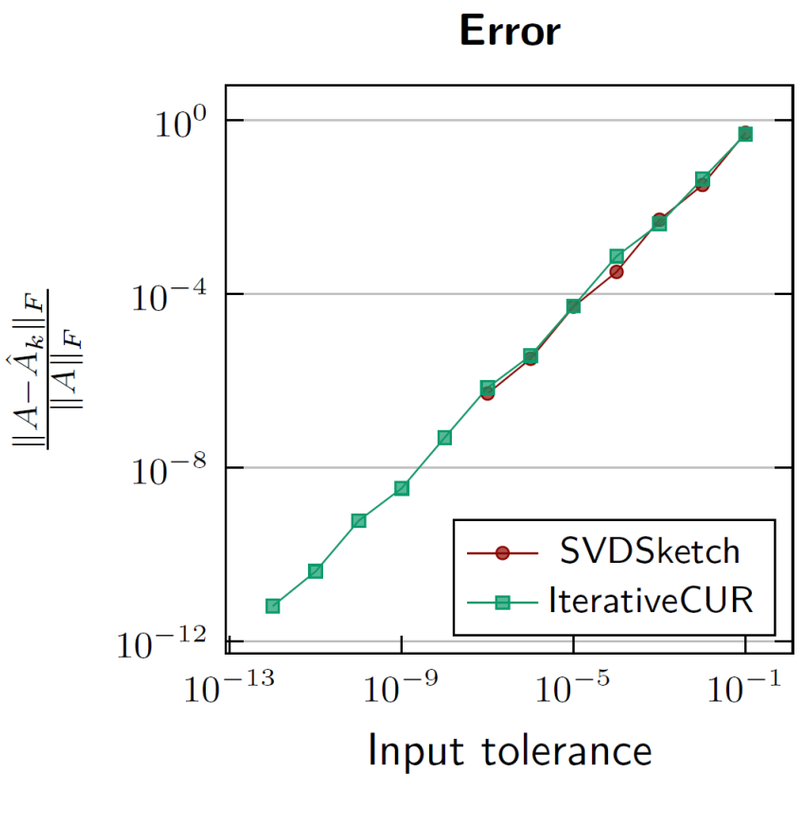
Recent work: IterativeCUR – numerical examples

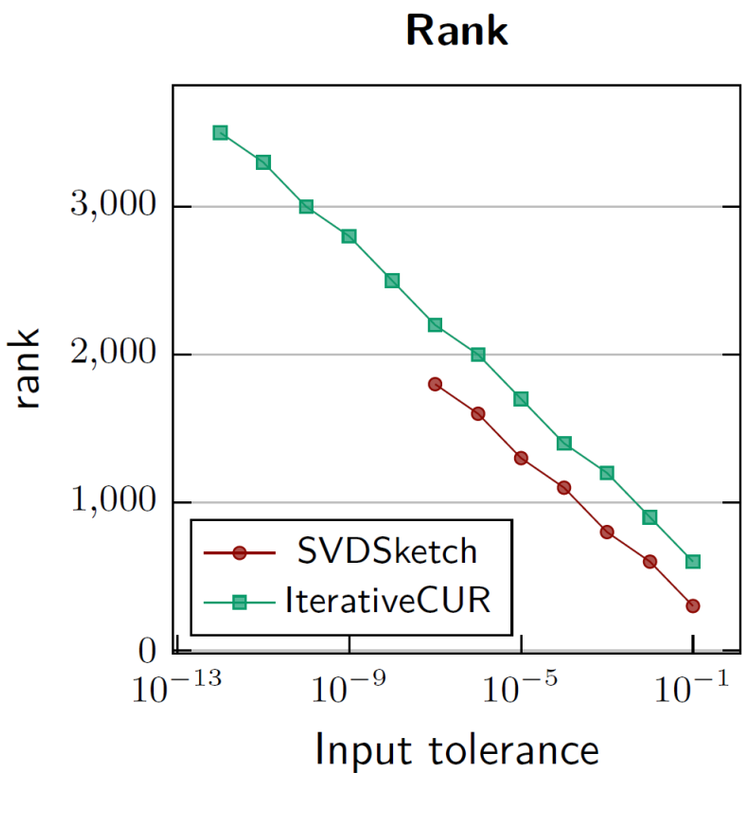


- SVDsketch: MATLAB's built-in [Yu-Yu-Li 18]
  - Roughly 'iterative RSVD' drawing new sketch each time
  - Cannot handle accuracy $\leq \epsilon_{\text{mach}}^{1/2} \approx 10^{-8}$
- s-LUPP: big sketch+LUPP
  - to test effect of reusing small sketch *G*
- IterativeCUR-LUPP: this work

Recent work: IterativeCUR – numerical examples







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

Outline

- Randomized low-rank approximation – “randomized SVD (RSVD)”.

  Well-established material. Background.

- A posteriori error estimation and rank adaptivity.

  Certified accuracy, adaptive rank selection, and automated codes.

- Data-sparse representations of global operators in scientific computing.

  Extending the RSVD methodology to PDE solution operators, DtNs, evolution operators, . . .

  Not globally low rank. “Hierarchical matrices”.

  Objective: Linear complexity direct solvers for challenging PDEs.

- If time permits:

  “Block nullification” and “randomized simultaneous compression and factorization”.

  How to actually execute the compression step for global operators.

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 linear 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").
- Techniques for *inverting, factorizing, exponentiating, ...* such operators exist, with progress ongoing — "$\mathcal{H}$-matrices", "Fast Direct Solvers", "inverse FMM" ...
- Today: Randomized methods for building data sparse representations.

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 linear 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 complex physical systems, and then *directly* solve the resulting system of equations.

Linear complexity direct solvers for many huge dense and sparse linear systems.

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 linear 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.

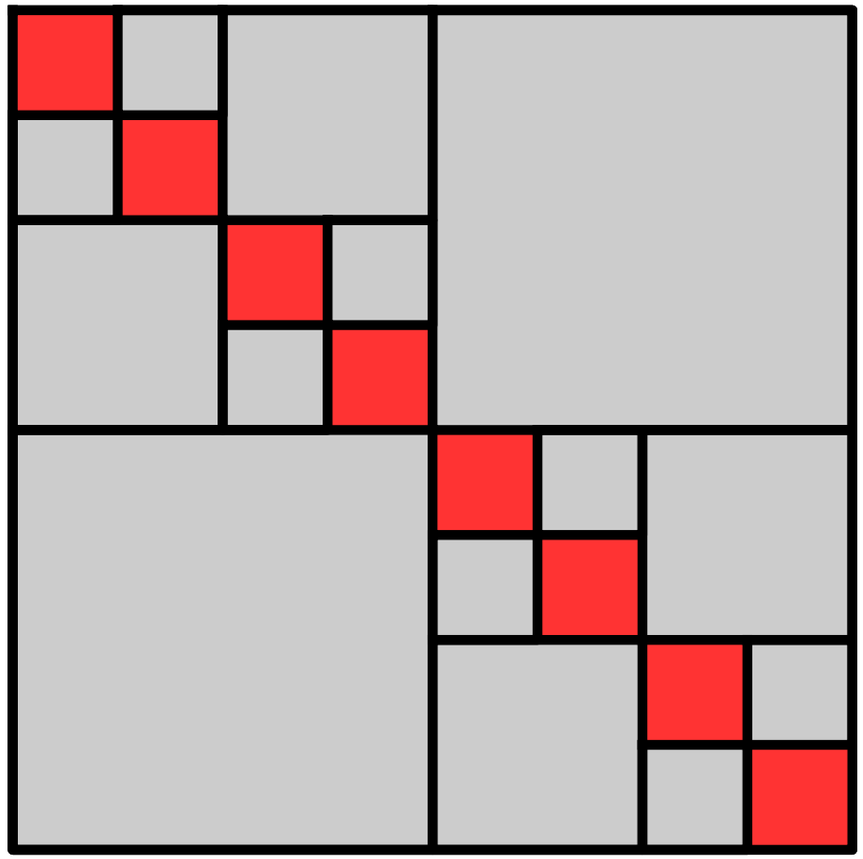
Key technical tool: "Rank structured matrices". ($\mathcal{H}$-matrices, HODLR, HSS, ...)

Many of the large dense matrices that arise can be tessellated into blocks in such a way that many large off-diagonal blocks have *numerically low rank*. Often, $O(N)$ complexity, or something close to it, is attainable.

Rank structured matrices (a.k.a. "hierarchical matrices", or "$\mathcal{H}$-matrices")

We use the term *rank structured* to refer to matrices that are not themselves of globally low rank, but can be tessellated into sub-blocks in such a way that each block is either small or of low numerical rank.

We focus on "hierarchical" tessellations (as the one shown on the right). Some techniques apply to "flat" formats as well.



All gray blocks have low rank.

Linear algebraic generalization of "Fast Multipole Method" for applying integral operators.

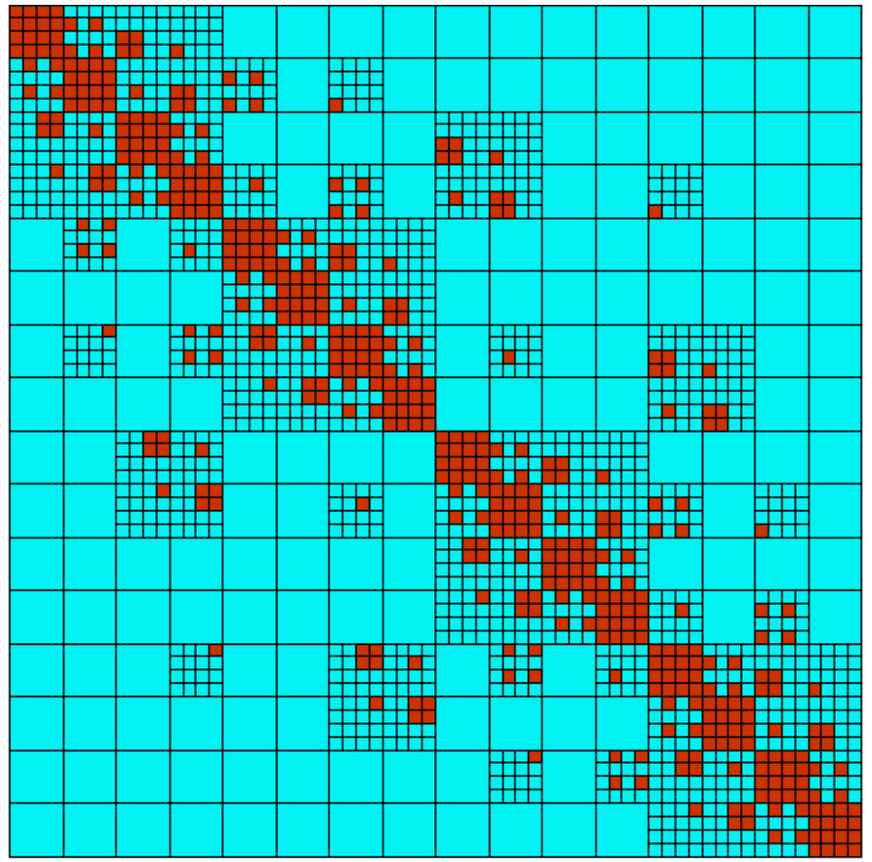
Hierarchically rank structured matrices often admit linear or close to linear complexity algorithms for the matrix-vector multiply, matrix-matrix multiply, LU factorization, etc.

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, . . .); . . .

Rank structured matrices (a.k.a. "hierarchical matrices", or "$\mathcal{H}$-matrices")

We use the term *rank structured* to refer to matrices that are not themselves of globally low rank, but can be tessellated into sub-blocks in such a way that each block is either small or of low numerical rank.

We focus on "hierarchical" tessellations (as the one shown on the right). Some techniques apply to "flat" formats as well.



Tessellations are often complicated.

Linear algebraic generalization of "Fast Multipole Method" for applying integral operators.

Hierarchically rank structured matrices often admit linear or close to linear complexity algorithms for the matrix-vector multiply, matrix-matrix multiply, LU factorization, etc.

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, . . .); . . .

Versions of rank structure:

There is a whole zoo of different rank-structured matrix "formats":

	Flat	Hierarchical	
		General bases	Nested bases
Weak admissibility	Block Separable	Hierarchically off-diagonal low rank (HODLR)	Hierarchically Block Separable (HBS/HSS); recursive skeletonization
Strong admissibility	Block Low Rank	$\mathcal{H}$-matrices; Barnes-Hut	$\mathcal{H}^2$-matrices; Fast Multipole Method; strong recursive skeletonization

Complexity of implementation *increases* as you go down and to the right in the table.

Asymptotic flop count *decreases* as you go down and to the right in the table.

The higher the dimension, the more complex scheme you need to use.

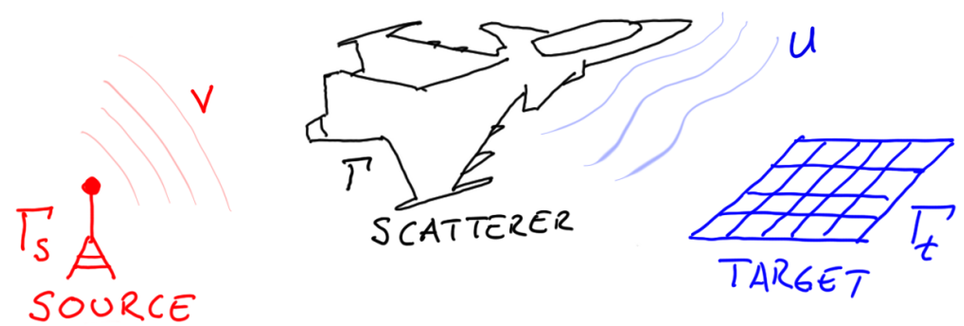
Recommendation: Use the simplest format that gives acceptable computational cost.

Note: *In principle, the term "$\mathcal{H}$-matrix" is extremely broad — every other format is a special case.*

However, the table reflects the standard usage of the term.

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

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

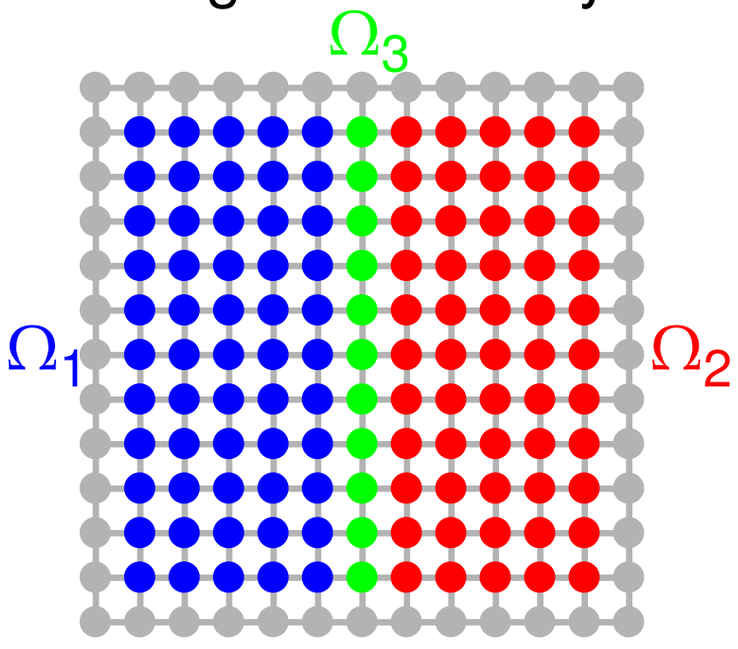
The integral equation (3) has several advantages over the PDE (2), including:

- 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.

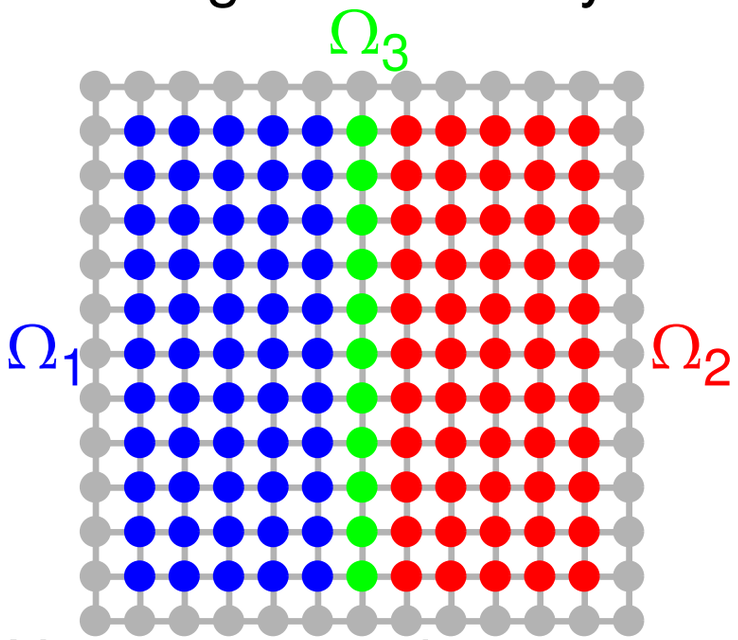
But both this matrix and its inverse are rank structured $\rightarrow O(N)$ solvers possible.

Example: 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:



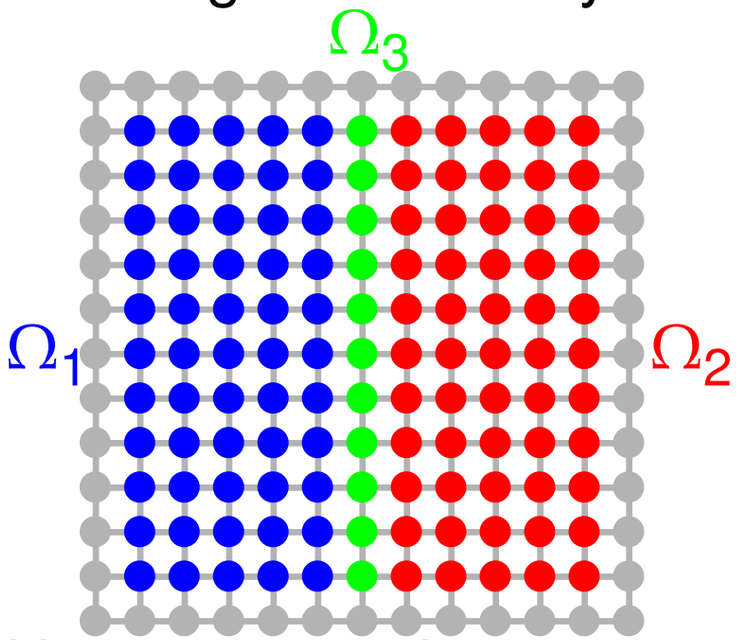
$$\mathbf{A} = \begin{array}{|c|c|c|} \hline \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \\ \hline \end{array}$$

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

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

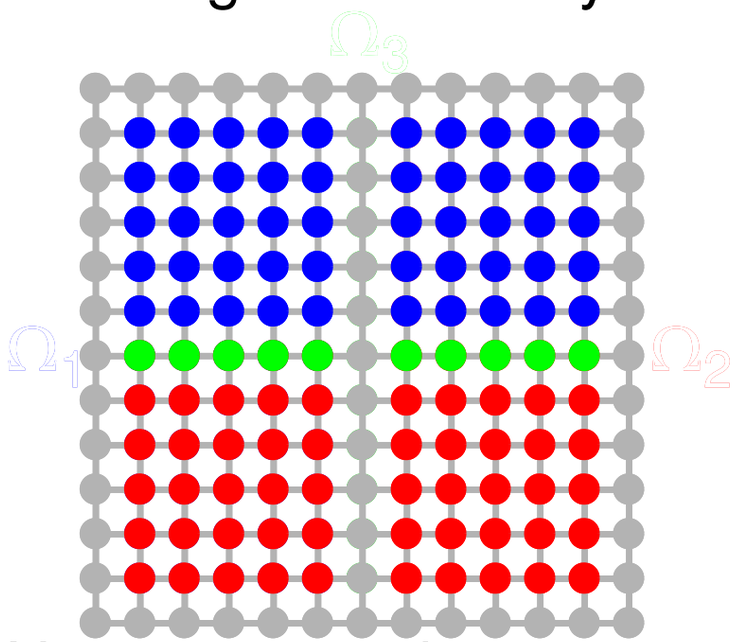
where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement.*

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

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

Notice the obvious recursion!

Example: Schur complements

Typically, nested dissection orderings are more complicated:

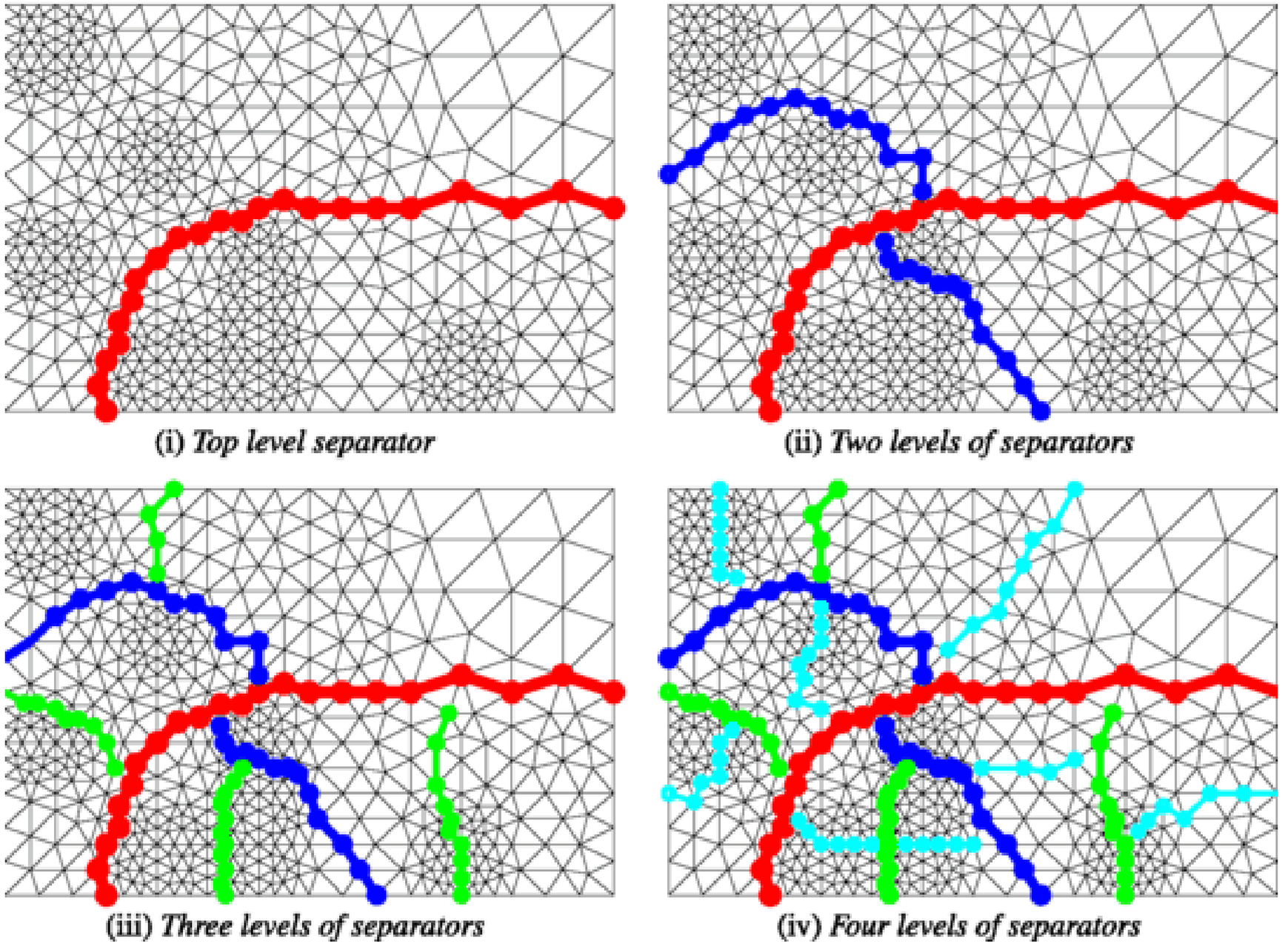


Image credit: Jianlin Xia, "Robust and Efficient Multifrontal Solver for Large Discretized PDEs", 2012

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 $O(N^2)$.

Iain taught me this in $\sim$1997!

Example: Schur complements

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

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

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

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

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

	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.

Black-box access model

Assume that we can apply the operator:

$$\mathbf{x} \mapsto \mathbf{A}\mathbf{x}, \qquad \mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}.$$

Can we recover a hierarchical/rank-structured representation from only a small number of global probes?

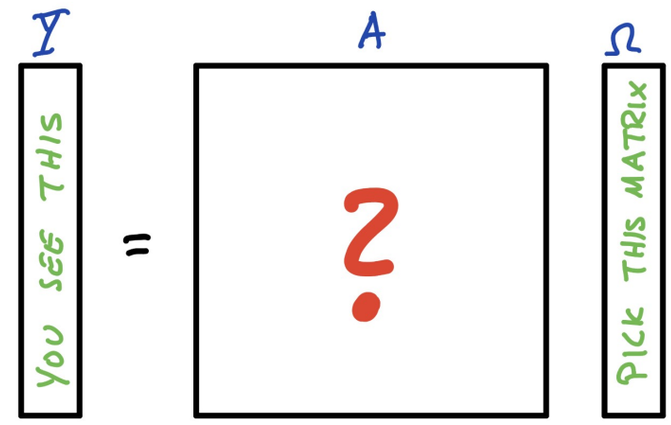
This is the rank-structured analogue of the RSVD.

Compression of a rank-structured matrix

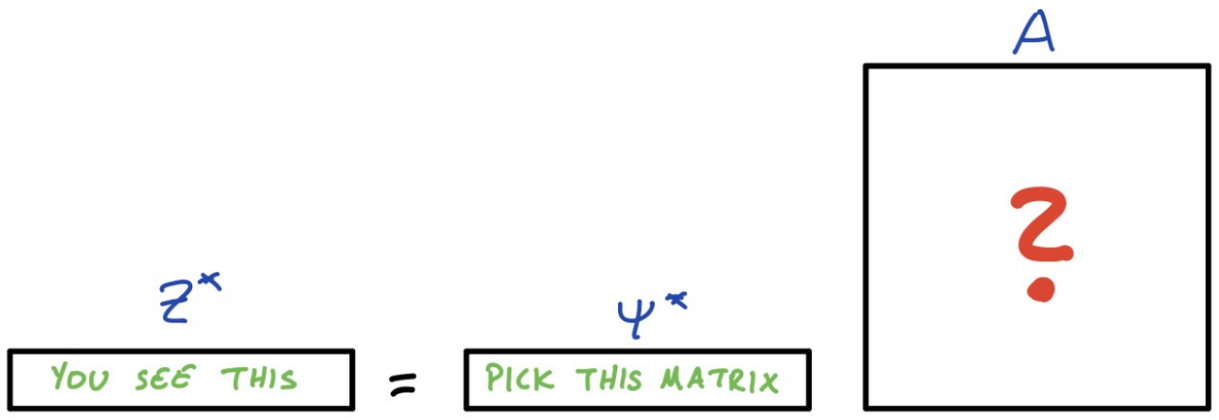
Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

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:



In the present context, the application of $\mathbf{A}$ to $\mathbf{\Omega}$ typically involves running a legacy PDE solver, or applying some known fast method for the matrix-vector multiplication.

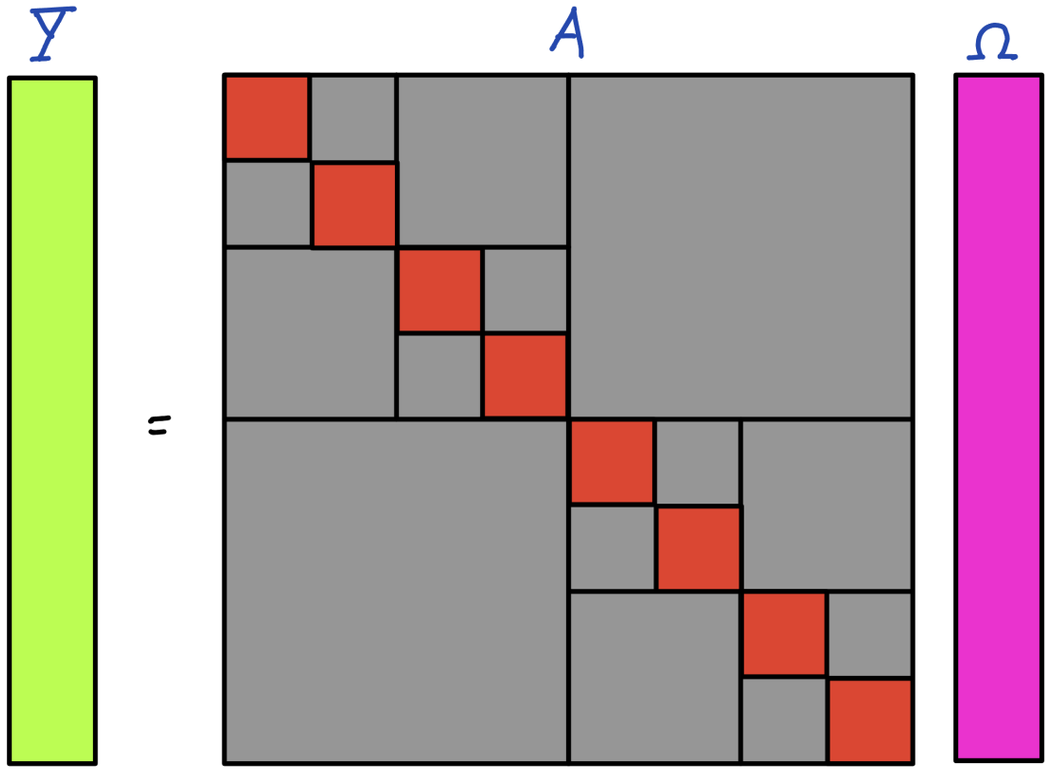
Compression of a rank-structured matrix

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 Randomized SVD from Part 1 solves the problem when $\mathbf{A}$ has globally low rank.

The challenge in the rank structured case is that each part of the matrix $\mathbf{Y}$ contains a mix of components from many off-diagonal blocks, and from some dense blocks too.



How do you disentangle these components?

Compression of a rank-structured matrix: How do you do it?

1. **With entry evaluation:** If you can evaluate individual entries of the target matrix, then you can simply subtract off the dense blocks from the sample matrix.
   - First algorithms of this nature to be developed (2008).
   - Leads to optimal complexity, and high practical speed.
   - Restrictive assumption! Excludes many important applications.

2. **With zero blocks in test matrix:** Create test matrices with strategically placed zero blocks that hit the dense blocks of the target matrix.
   - First complete "black box" algorithms (2011).
   - Every level must be processed independently $\rightarrow O(N \log N)$ complexity.
   - Pre-factors tend to be large. Can be reduced through graph coloring algorithms that design bespoke zero patterns for any given geometry.

3. **With algebraic structure in the test matrix:** It turns out to be possible to form test matrices that allow you to retro-actively go back and create zeros blocks where you like *after* extracting a universal sample.
   - High practical speed, and first method that is $O(N)$ and completely black box (2022).
   - Original method works best for 2D problems, or problems on surfaces in 3D.
   - Current work is aimed at finding different algebraic structures that lead to high efficiency for fully general 3D geometries.
   - It turns out to be possible to *invert* the matrix as you go!

Compression of a rank-structured matrix: How do you do it?

1. **With entry evaluation:** If you can evaluate individual entries of the target matrix, then you can simply subtract off the dense blocks from the sample matrix.

   *[P.G. Martinsson, SIMAX, **32**(4), 2011. Later improvements by Jianlin Xia, Xiaoye Sherry Li, . . .*]

2. **With zero blocks in test matrix:** Create test matrices with strategically placed zero blocks that hit the dense blocks of the target matrix.

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

   *J. Levitt & P.G. Martinsson, JCAM **451**(1), 2024.]*

3. **With algebraic structure in the test matrix:** It turns out to be possible to form test matrices that allow you to retro-actively go back and create zeros blocks where you like *after* extracting a universal sample.

   *[J. Levitt & P.G. Martinsson, SISC, **46**(3), 2024. K. Pearce, A. Yesypenko, J. Levitt, P.G. Martinsson, to appear in SIMAX*

   (arXiv:2501.05528). A. Yesypenko & P.G. Martinsson, to appear in J. Sci. Comp. (arxiv:2311.01451).]

4. **Plenty of related work recently!**

   Exploiting the continuum structure of the underlying operator: A. Townsend & N. Boullé (2022).

   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.

   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].*

Outline

- Randomized low-rank approximation – “randomized SVD (RSVD)”.

  Well-established material. Background.

- A posteriori error estimation and rank adaptivity.

  Certified accuracy, adaptive rank selection, and automated codes.

- Data-sparse representations of global operators in scientific computing.

  Extending the RSVD methodology to PDE solution operators, DtNs, evolution operators, . . .

  Not globally low rank. “Hierarchical matrices”.

  Objective: Linear complexity direct solvers for challenging PDEs.

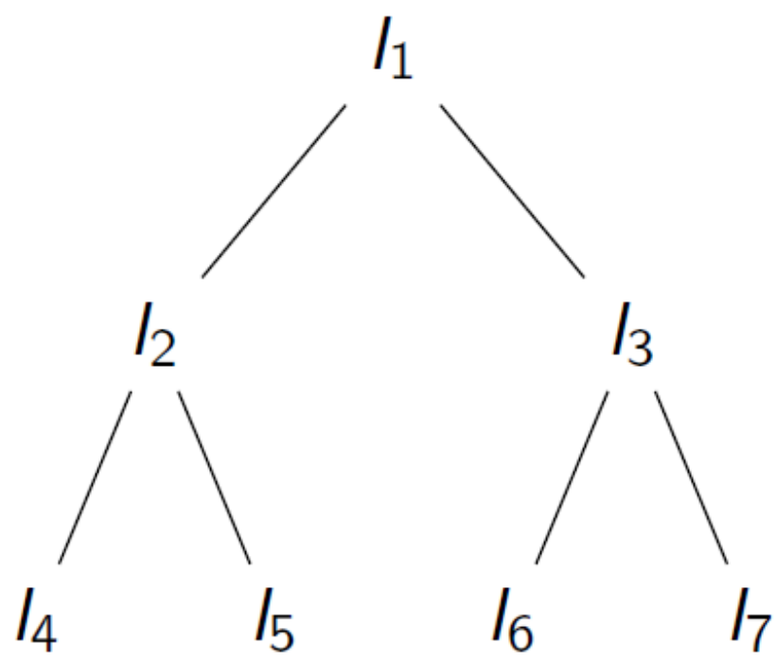
- If time permits:

  “Block nullification” and “randomized simultaneous compression and factorization”.

  How to actually execute the compression step for global operators.

Approximation of rank-structured matrices – A binary tree structure

An example of a binary tree structure for a matrix of size 400×400. The levels of the tree represent successively refined partitions of the index vector $[1, ..., 400]$.



Level 0: $I_1 = [1, 2, ..., 400]$

Level 1:
$I_2 = [1, ..., 200]$, $I_3 = [201, ..., 400]$

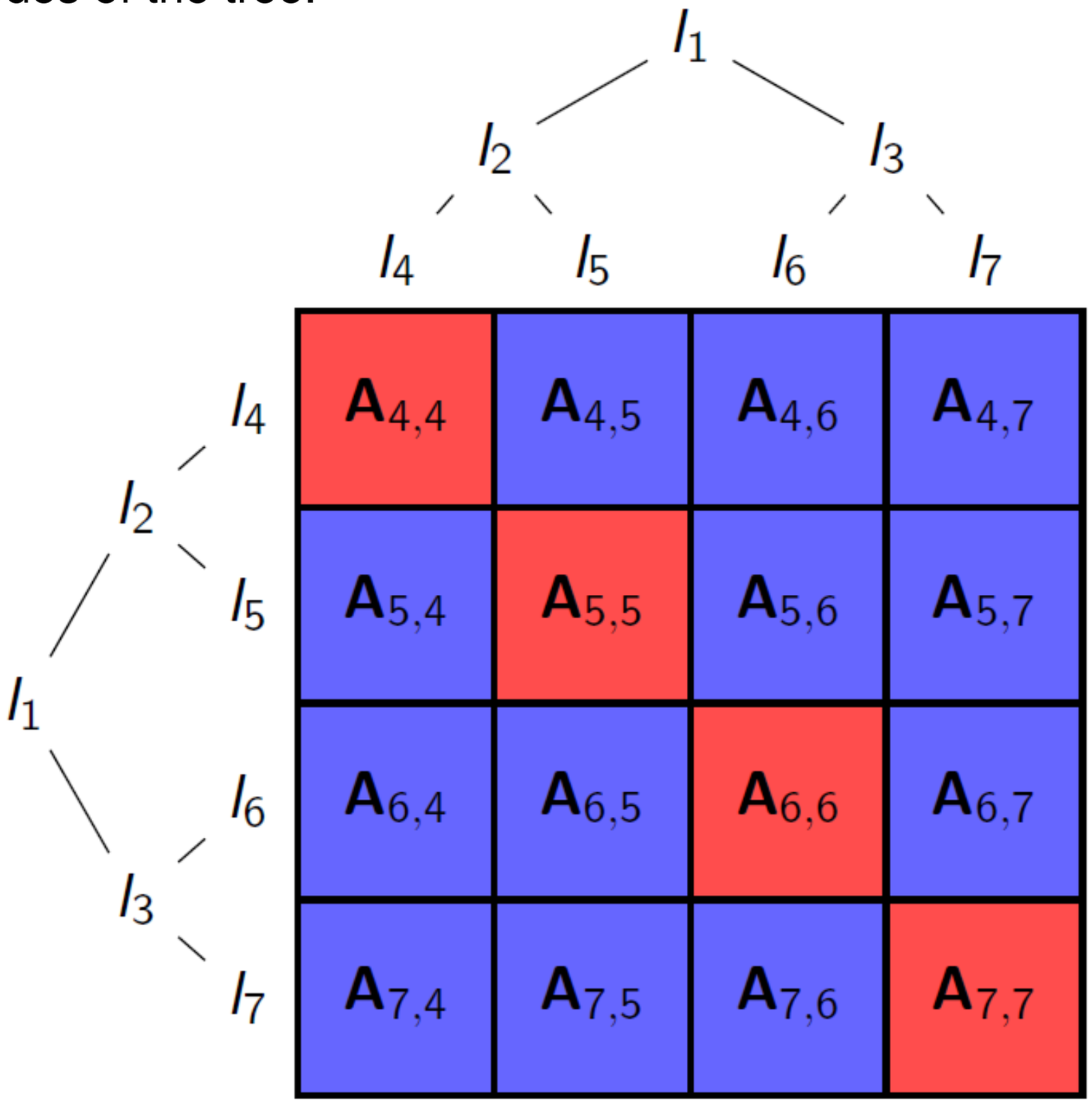
Level 2:
$I_4 = [1, ..., 100]$, $I_5 = [101, ..., 200]$,
$I_6 = [201, ..., 300]$, $I_7 = [301, ..., 400]$

Let m denote the leaf node size.

Let $L \approx \log(N/m)$ denote the depth of the tree.

Approximation of rank-structured matrices – HBS (a.k.a. HSS) structure

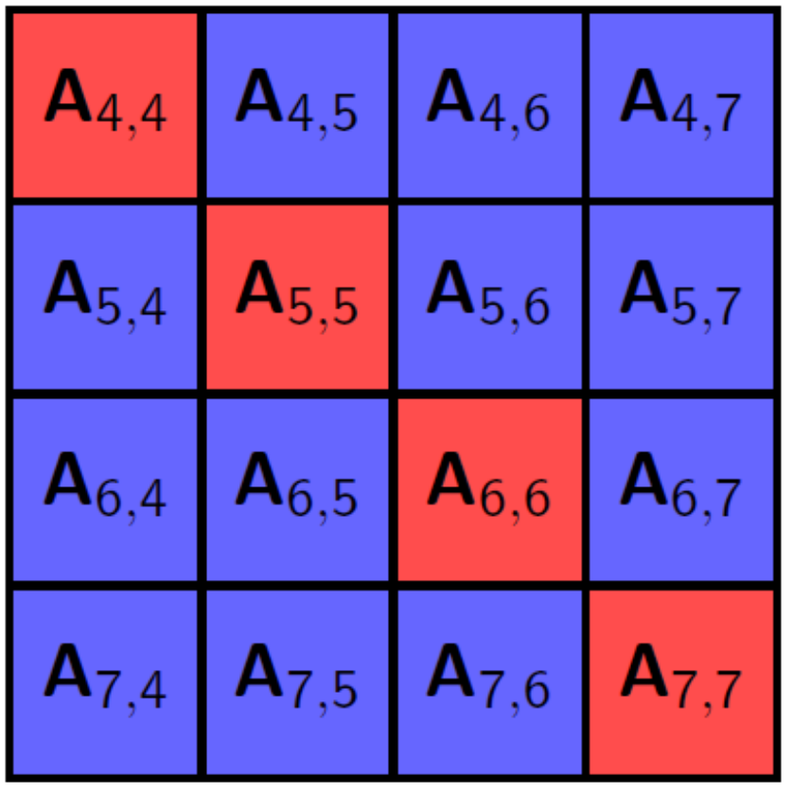
Consider the following tessellation of a matrix, where each block represents interactions between two leaf nodes of the tree.



Approximation of rank-structured matrices – HBS (a.k.a. HSS) structure

HBS requirements for the finest level: for every leaf node τ, there must exist basis matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ such that for every leaf node $\tau' \neq \tau$, we have

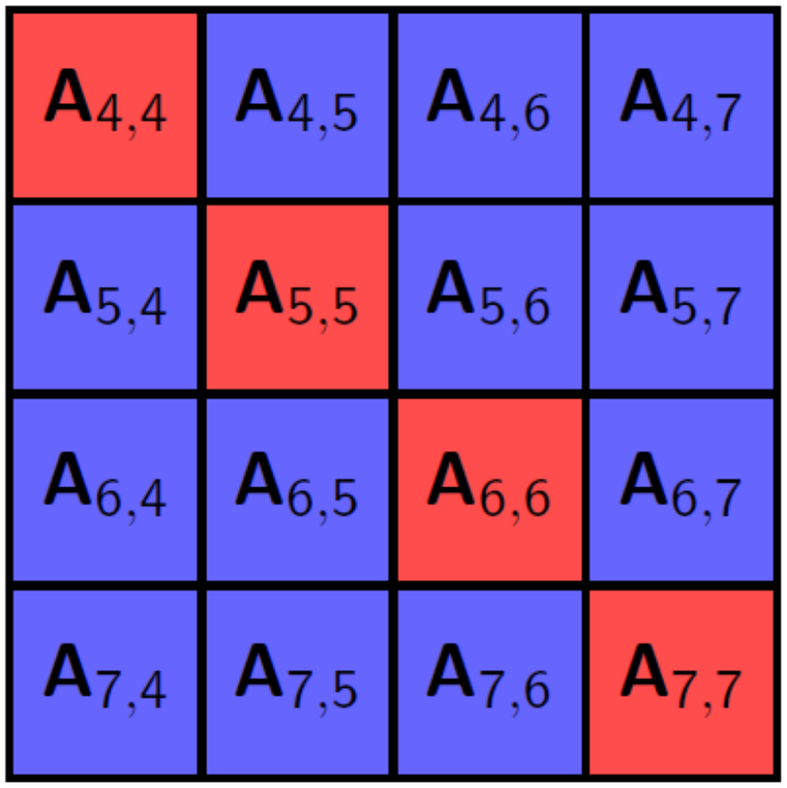
$$\begin{array}{ccccc} \mathbf{A}_{\tau,\tau'} & = & \mathbf{U}_\tau & \tilde{\mathbf{A}}_{\tau,\tau'} & \mathbf{V}_{\tau'}^*. \\ m \times m & & m \times k & k \times k & k \times m \end{array}$$



Approximation of rank-structured matrices – HBS (a.k.a. HSS) structure

HBS requirements for the finest level: for every leaf node τ, there must exist basis matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ such that for every leaf node $\tau' \neq \tau$, we have

$$\begin{array}{ccccc} \mathbf{A}_{\tau,\tau'} & = & \mathbf{U}_\tau & \tilde{\mathbf{A}}_{\tau,\tau'} & \mathbf{V}_{\tau'}^*. \\ m \times m & & m \times k & k \times k & k \times m \end{array}$$



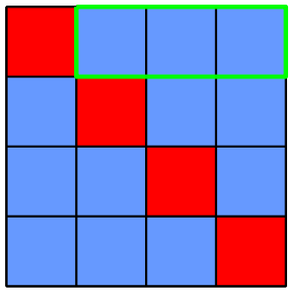
The on-diagonal blocks are not assumed to be low-rank, and pose the main challenge for black-box compression.

Question: How would you construct $\mathbf{U}_4$?

Approximation of rank-structured matrices – A naive approach

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

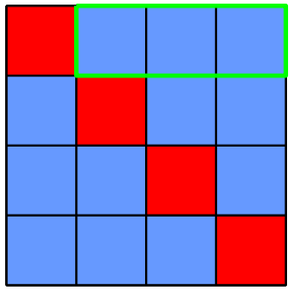
We seek a sample of $\mathbf{A}(I_4, I_4^c)$, the HBS row block of node 4.



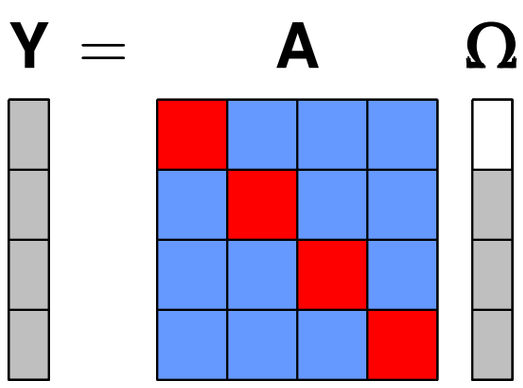
Approximation of rank-structured matrices – A naive approach

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.



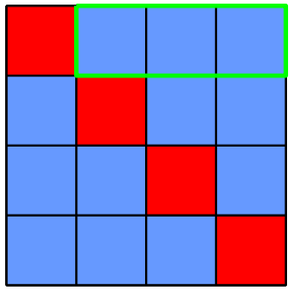
The naive approach is to sample with a random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4. Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^c)$.



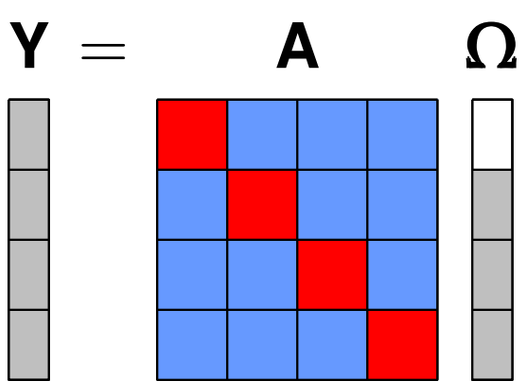
Approximation of rank-structured matrices – A naive approach

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 $\mathbf{A}(I_4, I_4^c)$.

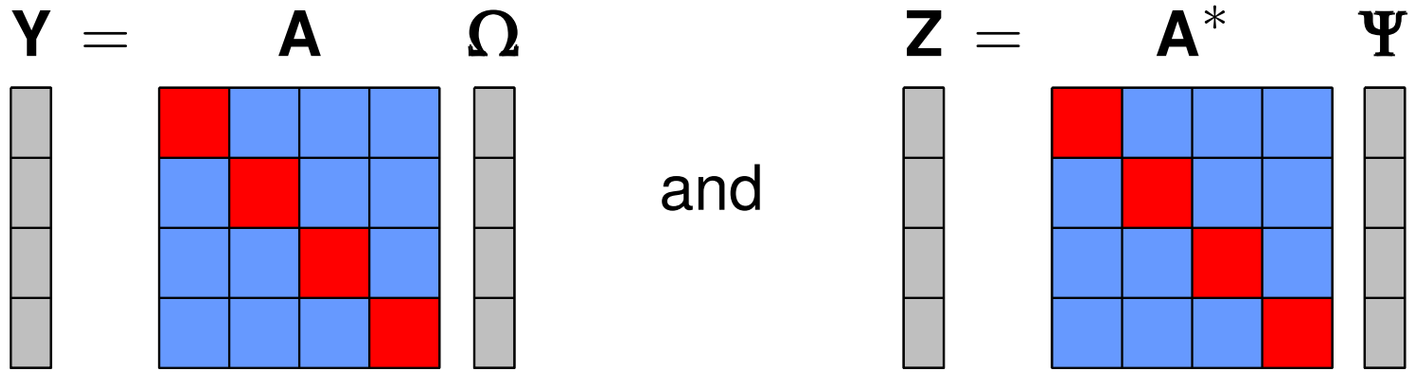


This scheme requires a bespoke test matrix *for each leaf node*, for a total of $\sim rN/m$ samples.

There is a lot of wasted information in $\mathbf{Y}$.

Approximation of rank-structured matrices – the "almost" black-box case

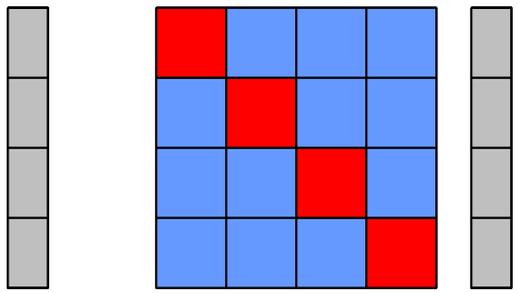
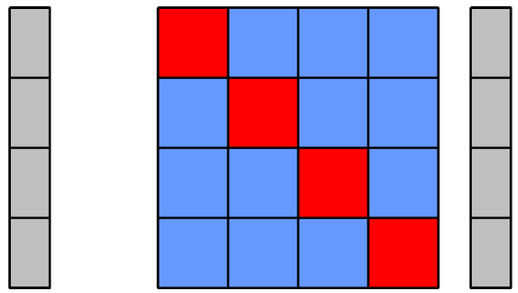
Sample $\mathbf{A}$ with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N\times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N\times r}$:



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

Approximation of rank-structured matrices – the "almost" black-box case

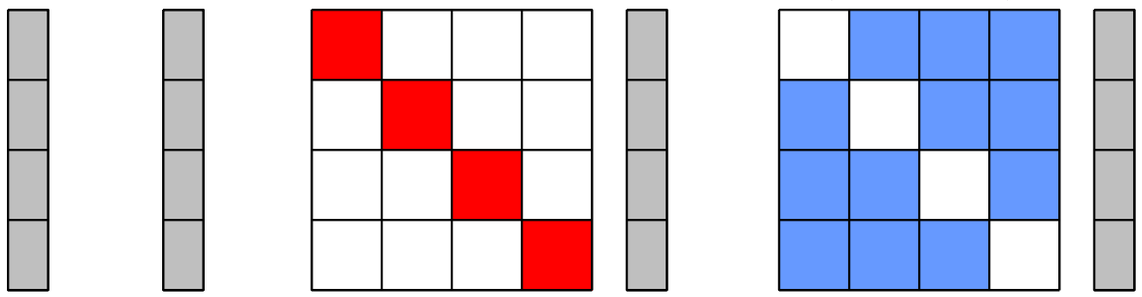
Sample $\mathbf{A}$ with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:

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

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

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

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



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

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

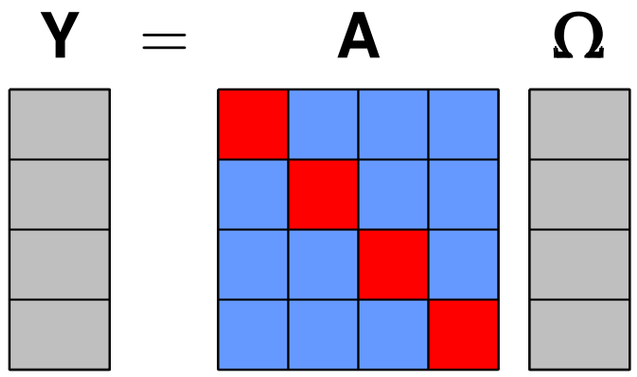
for *some* small matrices $\mathbf{B}_{i,j}$.

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

*[P.G. Martinsson, SISC, **32**(4), 2011]*

Approximation of rank-structured matrices – fully black box

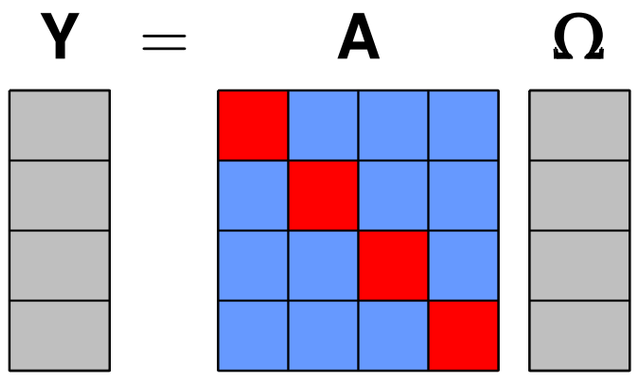
Sample $\mathbf{A}$ with a completely dense random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times (r+m)}$, where m is the leaf node size. (Think $m \approx 2k$ and $r = k + 10$.)



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

Approximation of rank-structured matrices – 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 $\mathbf{A}(I_4, I_4^c)$.

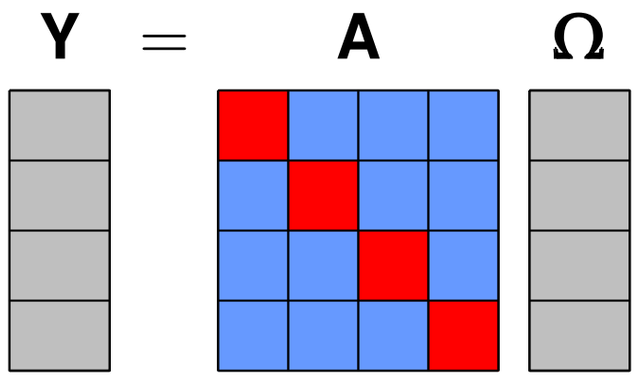
Since $\mathbf{\Omega}(I_4, :\ )$ is of size $m \times (r+m)$, it has a nullspace of dimension at least r. Let

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

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

Approximation of rank-structured matrices – 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 $r = k + 10$.)

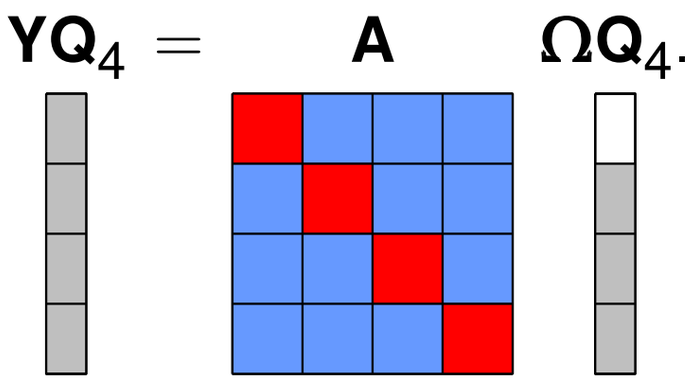


Let us consider the problem of finding a basis matrix $\mathbf{U}_4$ for the block $\mathbf{A}(I_4, I_4^c)$.

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 $\mathbf{\Omega}(I_4, :)$.Then



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

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

Approximation of rank-structured matrices – fully black box

All boxes on the same level can be processed in a similar way.

Once all the off-diagonal blocks have been compressed, it is possible to explicitly extract the required information about the diagonal blocks.

The scheme then proceeds to the next coarser level.
The downdating technique described earlier is used to obtain a sample of the “residual” matrix of roughly size $N/2 \times N/2$.
Use the same process that we used at the finest level to compress.

Then downdate, and proceed to process the matrix of size $N/4 \times N/4$.

Etc.

Next, a method that simultaneously compresses and factorizes the matrix.

Work in progress …

Randomized simultaneous factorization and compression (RSFC)

Let $\mathbf{A}$ be a dense rank structured matrix.

Reorder $\mathbf{A}$ to form a matrix $\mathbf{A}(I, I)$ that has contiguous off-diagonal blocks of low rank.

We will drive $\mathbf{A}(I, I)$ to block diagonal form through a sequence of factorizations

$$\begin{aligned}\mathbf{A}(I, I) &= \mathbf{G}_1 \mathbf{A}_1 \mathbf{H}_1, \\ \mathbf{A}_1 &= \mathbf{G}_2 \mathbf{A}_2 \mathbf{H}_2, \\ \mathbf{A}_2 &= \mathbf{G}_3 \mathbf{A}_3 \mathbf{H}_3,\end{aligned}$$

and so on until you obtain a *block diagonal* matrix $\mathbf{A}_M$ such that

$$\mathbf{A}(I, I) = \mathbf{G}_1 \mathbf{G}_2 \cdots \mathbf{G}_M \mathbf{A}_M \mathbf{H}_M \cdots \mathbf{H}_2 \mathbf{H}_1.$$

Each matrix $\{\mathbf{G}_i\}_{i=1}^M$ and $\{\mathbf{H}_i\}_{i=1}^M$ is the identity matrix with a dense block of size $O(k) \times O(k)$ somewhere. Each such matrix can be applied to a vector in cost $O(k^2)$.

We use the randomized procedure described to find the factor matrices. Starting with a universal sample

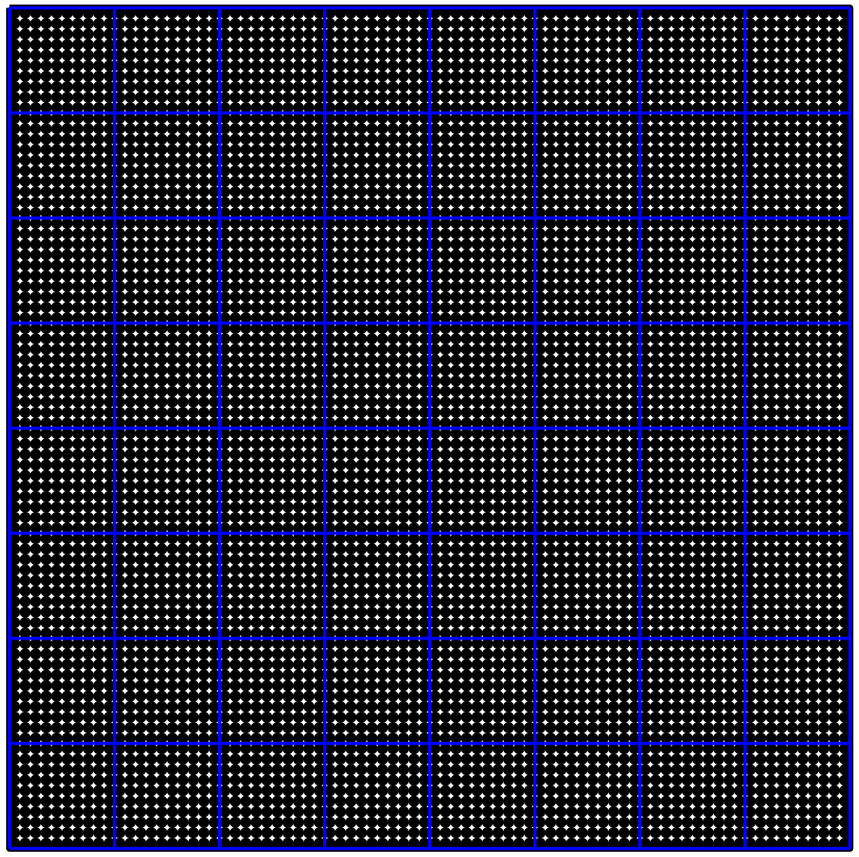
$$\mathbf{Y} = \mathbf{A}(I, I)\mathbf{\Omega},$$

we update the factorization after each transform

$$\underbrace{\mathbf{G}_1^{-1}\mathbf{Y}}_{=:\mathbf{Y}_1} = \underbrace{\mathbf{G}_1^{-1}\mathbf{A}(I, I)\mathbf{H}_1^{-1}}_{=:\mathbf{A}_1} \underbrace{\mathbf{H}_1\mathbf{\Omega}}_{=:\mathbf{\Omega}_1}.$$

Randomized simultaneous factorization and compression (RSFC)

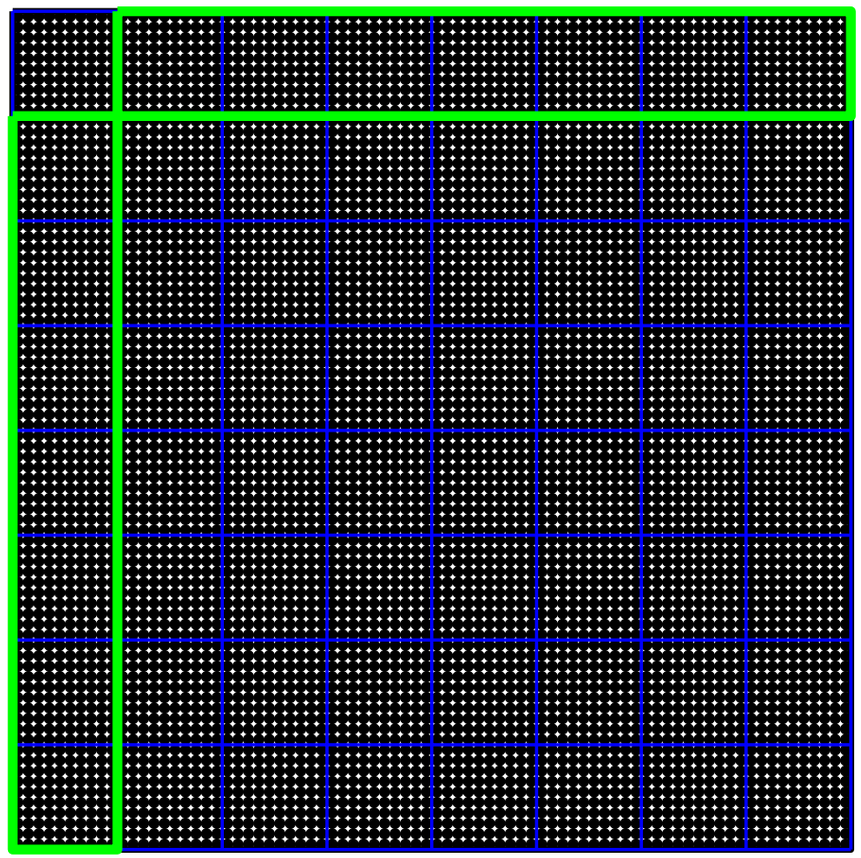
As an example, let us consider an 8×8 block matrix.



The original matrix.

Randomized simultaneous factorization and compression (RSFC)

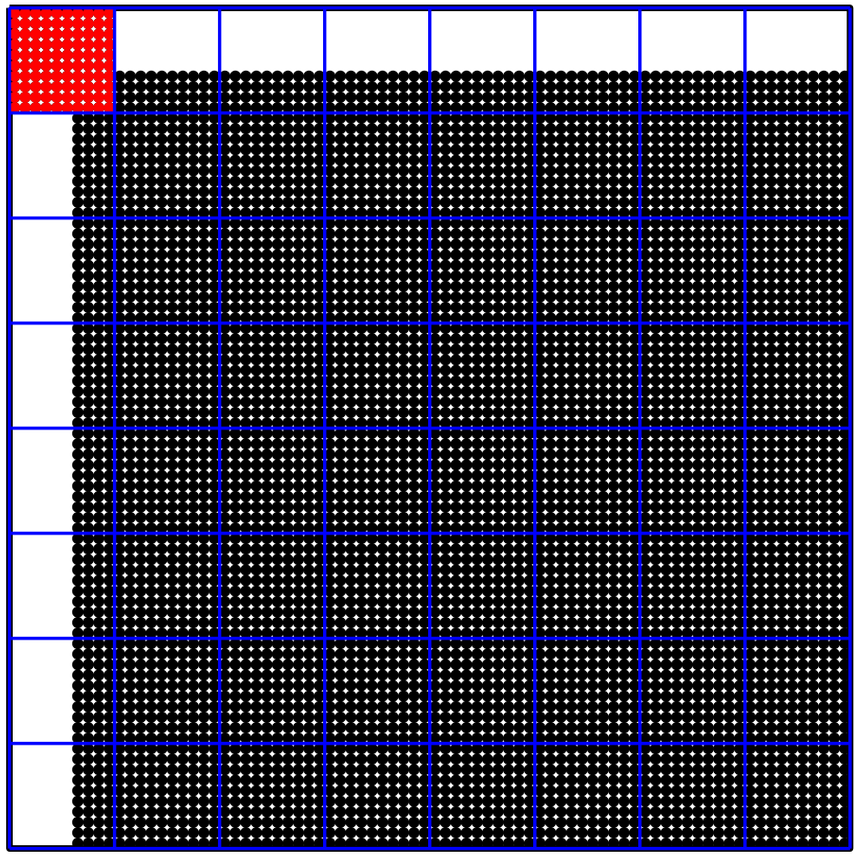
As an example, let us consider an 8×8 block matrix.



To start, execute the process described earlier ("block nullification") to find bases matrices for the green blocks (which are low rank).
Then apply unitary matrices to the first block row and the first block column to introduce zeros in the off-diagonal blocks.

Randomized simultaneous factorization and compression (RSFC)

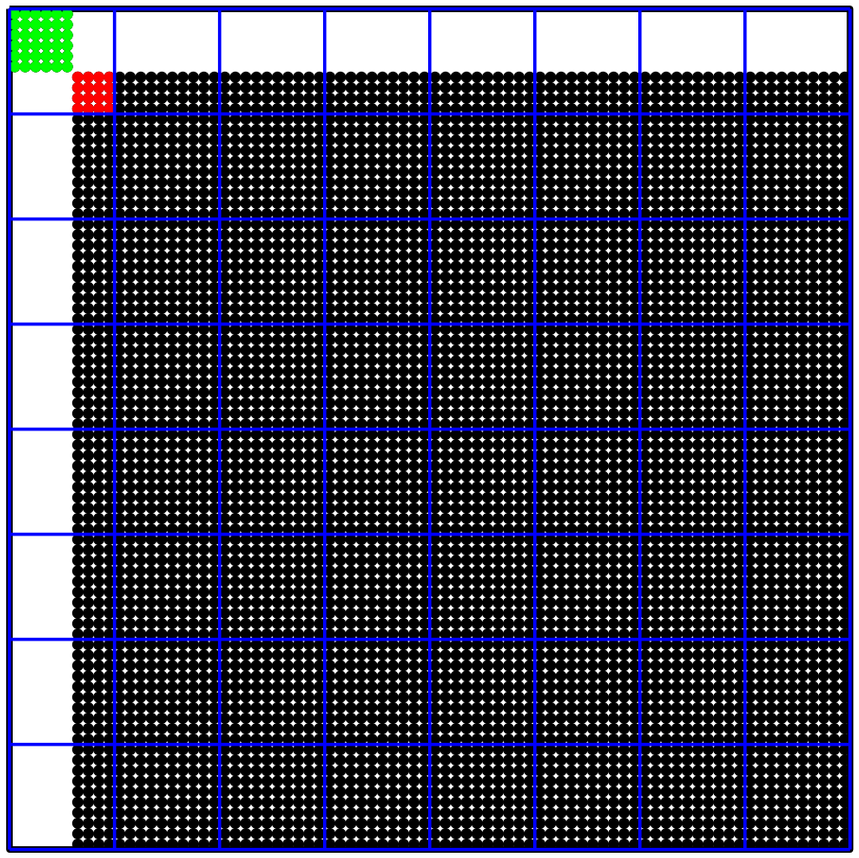
As an example, let us consider an 8×8 block matrix.



To start, execute the process described earlier ("block nullification") to find bases matrices for the green blocks (which are low rank).
Then apply unitary matrices to the first block row and the first block column to introduce zeros in the off-diagonal blocks.
(Alternatively, block Gaussian elimination could be used. Then all black entries would be untouched. But less numerically stable.)

Randomized simultaneous factorization and compression (RSFC)

As an example, let us consider an 8×8 block matrix.

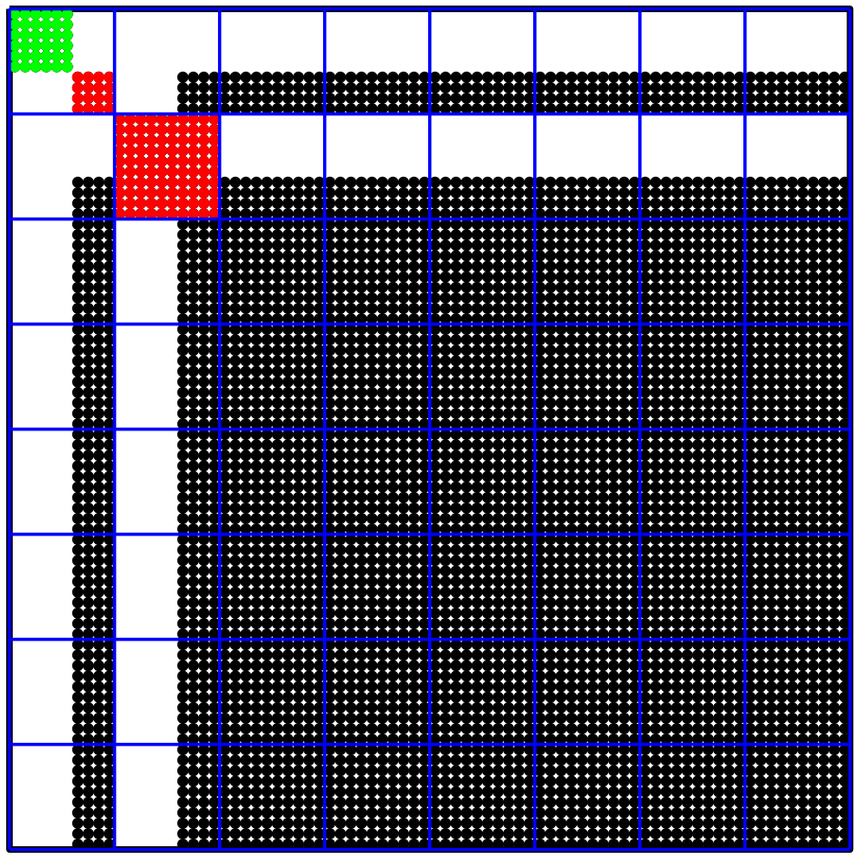


In this step, we performed block Gaussian elimination to introduce two zero blocks in the diagonal block. Note that this was a purely local operator that impacted only the nodes in the diagonal block.

Note: *The green block is now disconnected!*

Randomized simultaneous factorization and compression (RSFC)

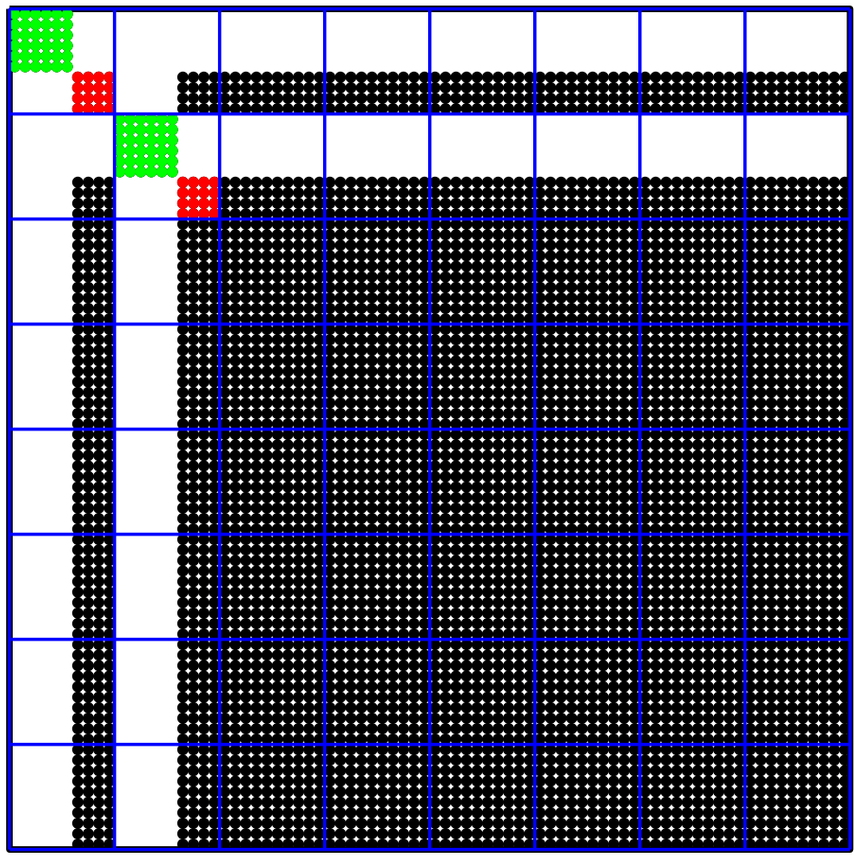
As an example, let us consider an 8×8 block matrix.



Process the next block the same way.

Randomized simultaneous factorization and compression (RSFC)

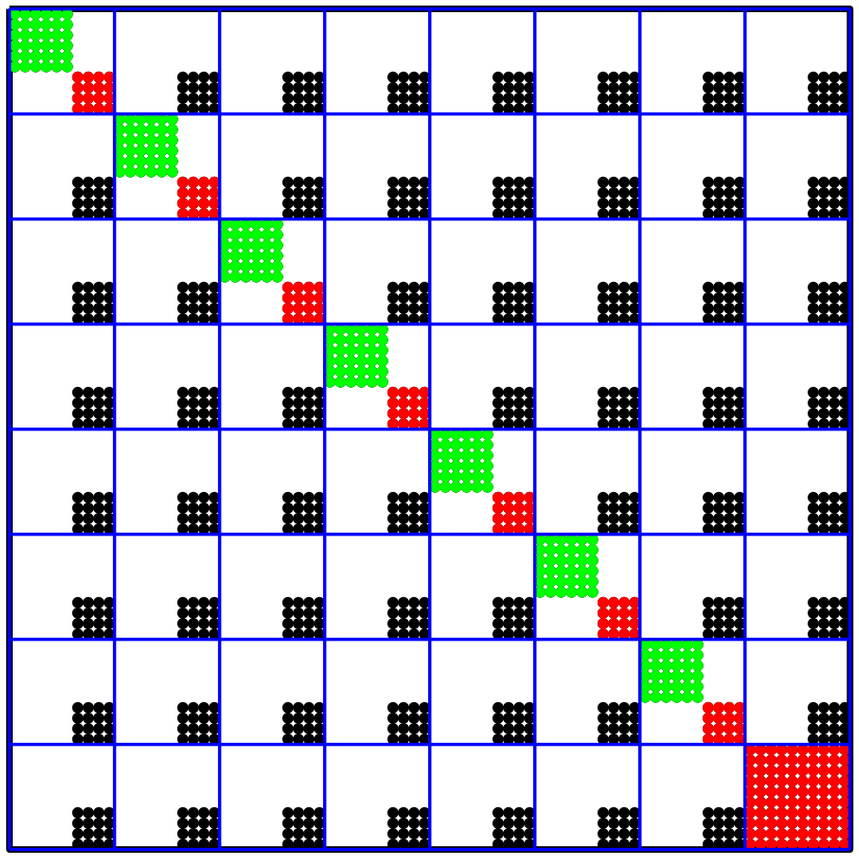
As an example, let us consider an 8×8 block matrix.



Process the next block the same way.

Randomized simultaneous factorization and compression (RSFC)

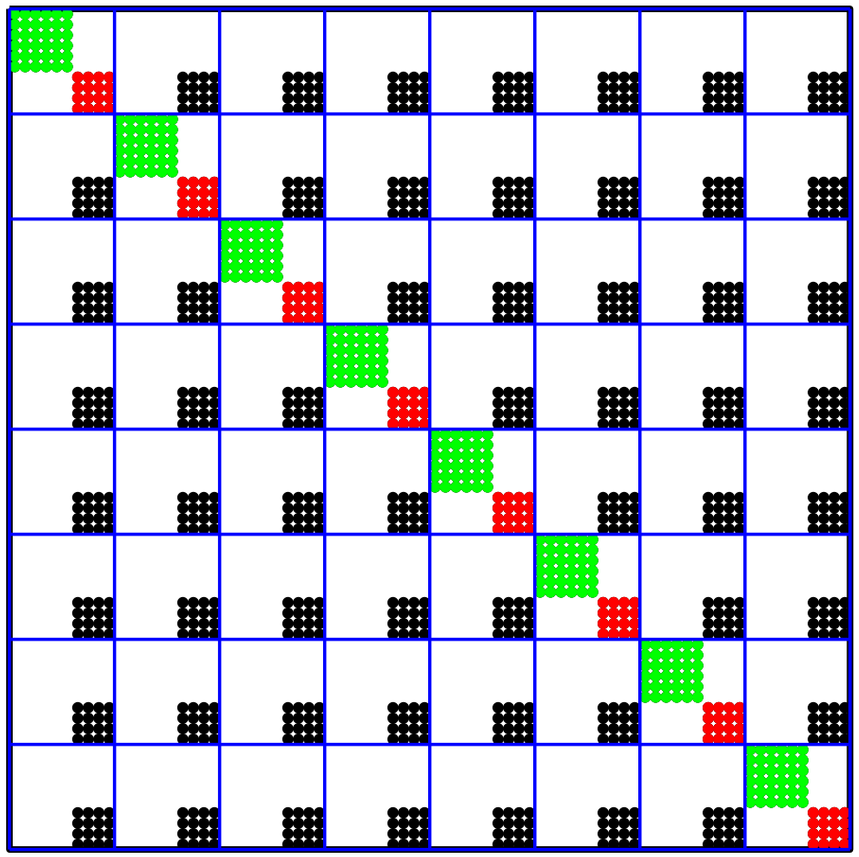
As an example, let us consider an 8×8 block matrix.



Continue through all blocks.

Randomized simultaneous factorization and compression (RSFC)

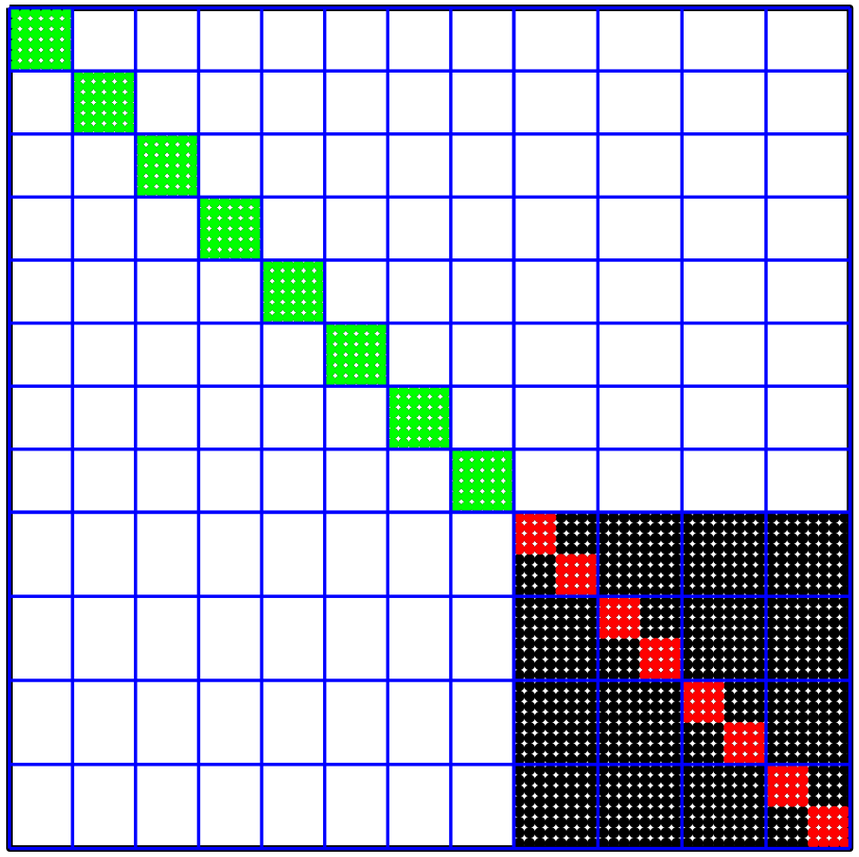
As an example, let us consider an 8×8 block matrix.



Continue through all blocks.

Randomized simultaneous factorization and compression (RSFC)

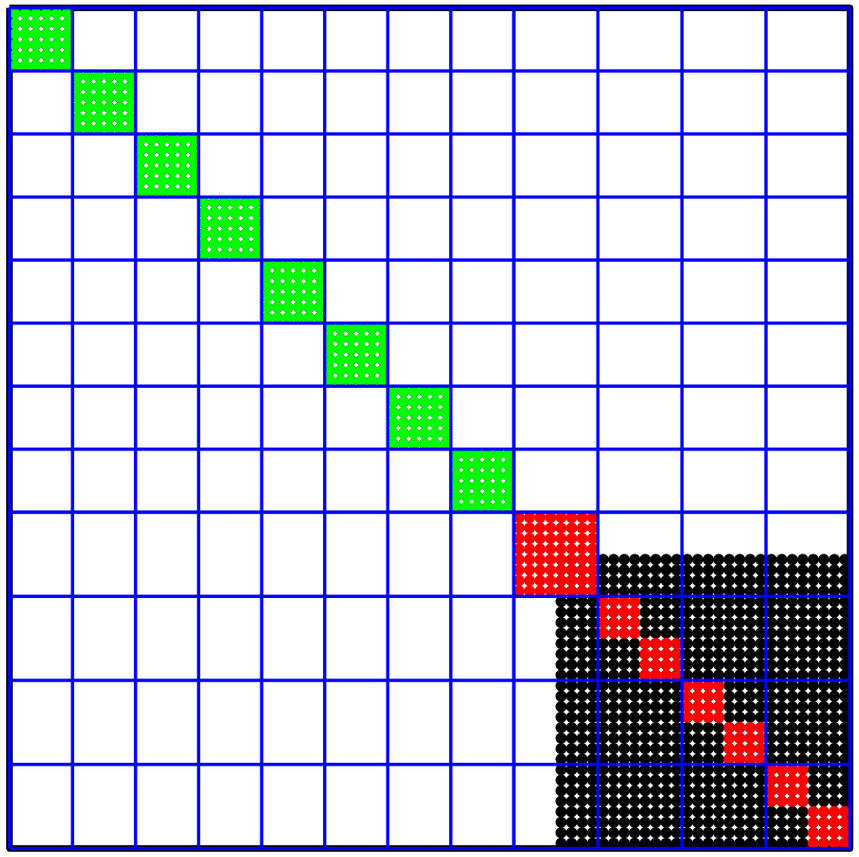
As an example, let us consider an 8×8 block matrix.



Now permute the indices so that the disconnected dofs are listed first.

Randomized simultaneous factorization and compression (RSFC)

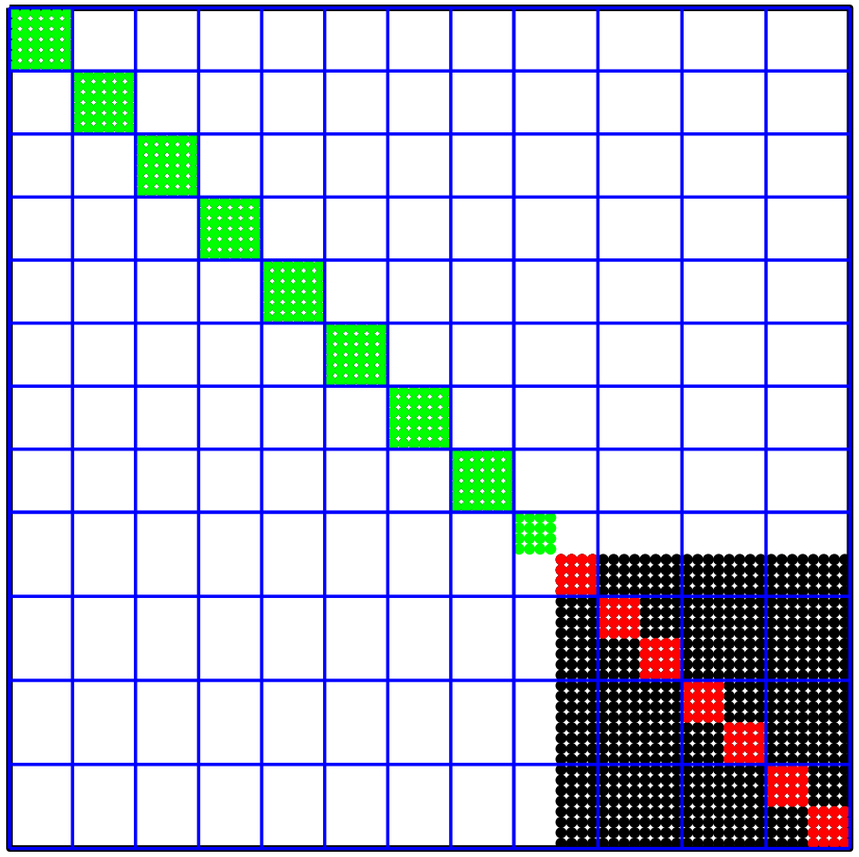
As an example, let us consider an 8×8 block matrix.



Proceed to compress the smaller matrix.

Randomized simultaneous factorization and compression (RSFC)

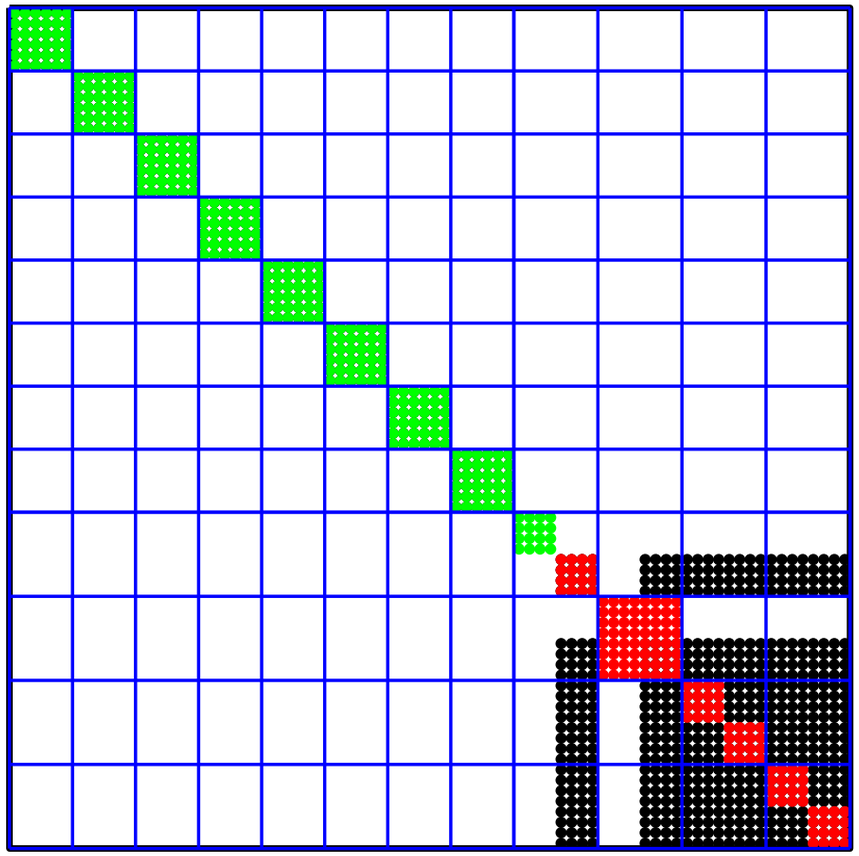
As an example, let us consider an 8×8 block matrix.



Proceed to compress the smaller matrix.

Randomized simultaneous factorization and compression (RSFC)

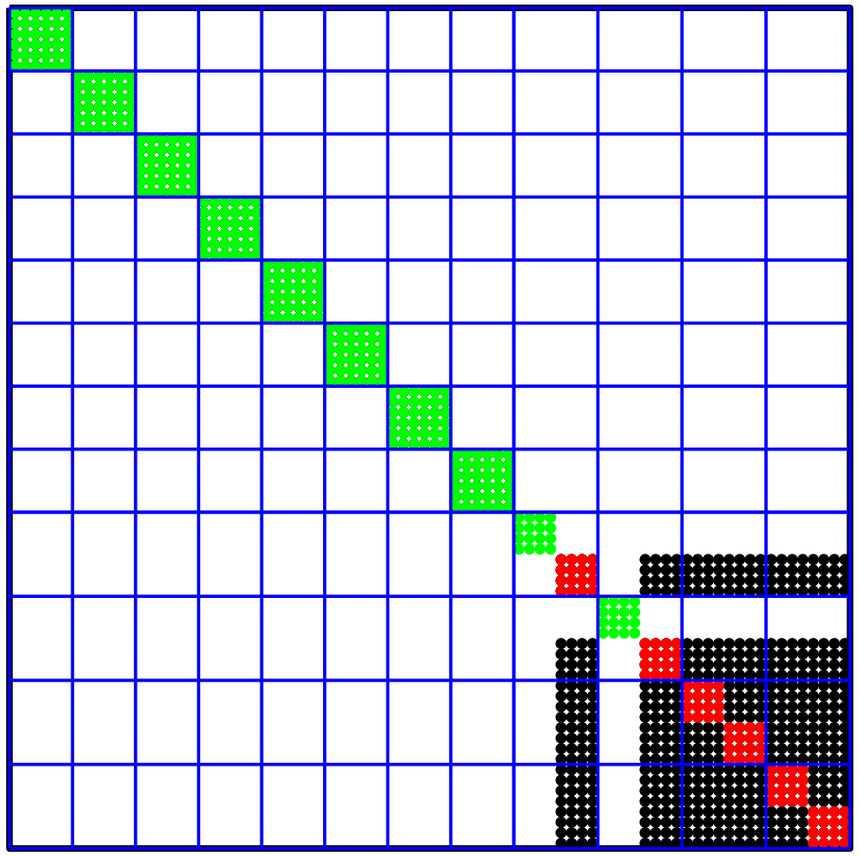
As an example, let us consider an 8×8 block matrix.



Proceed to compress the smaller matrix.

Randomized simultaneous factorization and compression (RSFC)

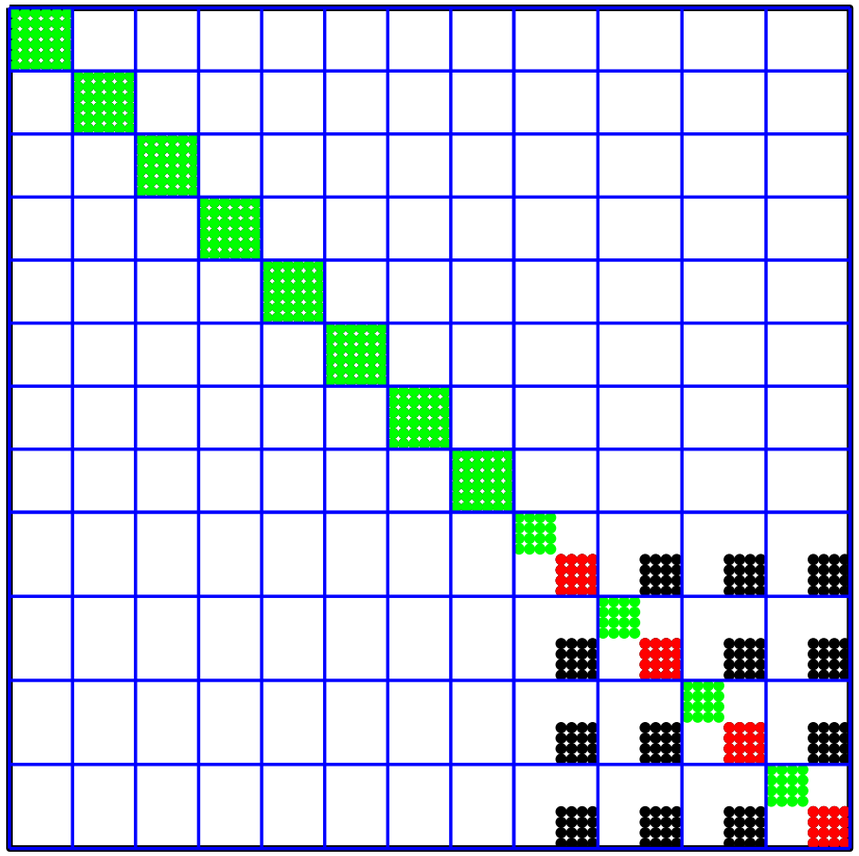
As an example, let us consider an 8×8 block matrix.



Proceed to compress the smaller matrix.

Randomized simultaneous factorization and compression (RSFC)

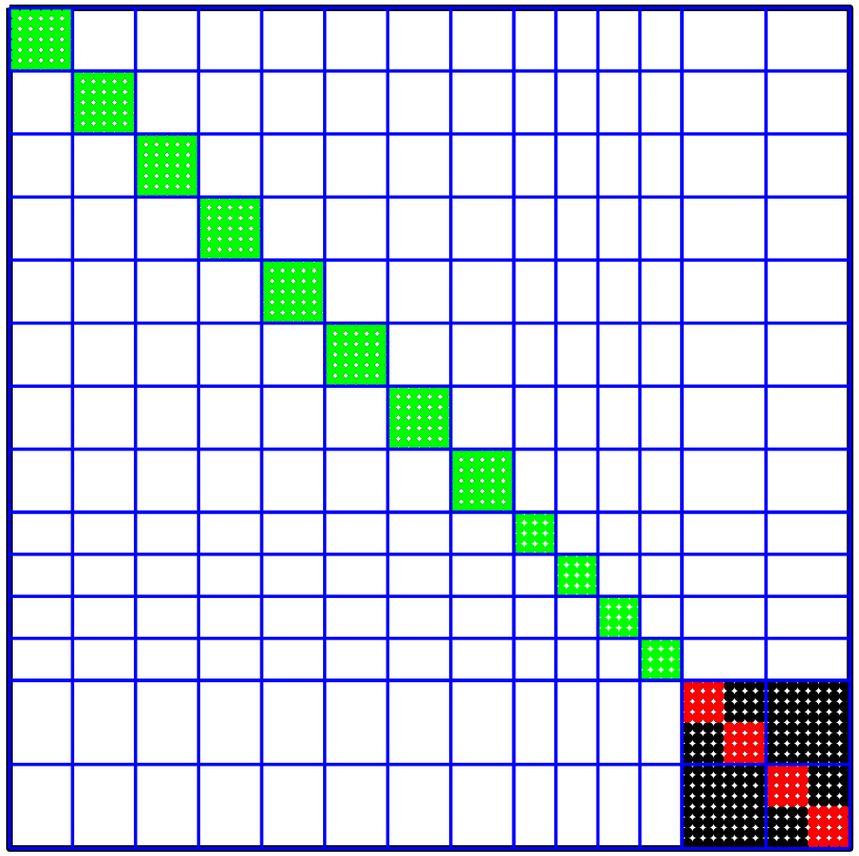
As an example, let us consider an 8×8 block matrix.



Proceed to compress the smaller matrix.

Randomized simultaneous factorization and compression (RSFC)

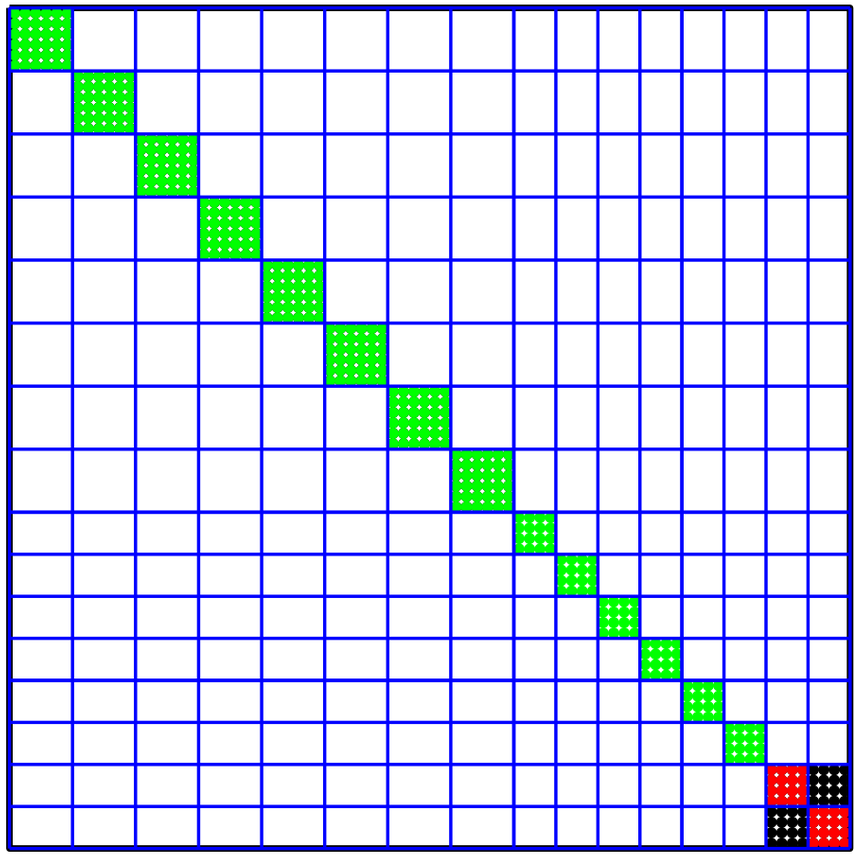
As an example, let us consider an 8×8 block matrix.



Permute.

Randomized simultaneous factorization and compression (RSFC)

As an example, let us consider an 8×8 block matrix.



Compress again, then permute.

Randomized simultaneous factorization and compression (RSFC)

Let us summarize the RSFC process:

- The algorithm proceeds through all nodes in the tree, visiting each node once.
- The output is a direct representation of both $\mathbf{A}$ and of $\mathbf{A}^{-1}$.
- At each step, a set of degrees of freedom are uncoupled from the global system by introducing zeros in the coefficient matrix.
- *Whatever transform is in principle applied to* $\mathbf{A}$*, is in actuality only applied to the test and the sample matrices.*

- Each step requires $O(k^3)$ flops, and $O(k^2)$ storage.
  Overall cost is $O(Nk^2)$ flops and $O(Nk)$ storage.
- All blocks on one level can be processed in parallel.
  $\rightarrow$ Ideal for batched linear algebra on GPUs.
- Two choices for transform matrix for the off-diagonal blocks:
  Unitary: Most numerically stable.
  Block Gaussian ("skeletonization"): Surviving off-diagonal entries are unchanged.

Randomized simultaneous factorization and compression (RSFC)

Vulnerability:

Randomized simultaneous factorization and compression (RSFC)

Vulnerability: Recall that RSFC requires repeated updating of the test matrix

$$\boldsymbol{\Omega} \quad \mapsto \quad \boldsymbol{\Omega}_1 = \mathbf{H}_1\boldsymbol{\Omega} \quad \mapsto \quad \boldsymbol{\Omega}_2 = \mathbf{H}_2\boldsymbol{\Omega}_1 \quad \mapsto \quad \boldsymbol{\Omega}_3 = \mathbf{H}_3\boldsymbol{\Omega}_2 \cdots$$

An issue arises in that $\mathbf{H}_1$ depends on $\boldsymbol{\Omega}$, so $\boldsymbol{\Omega}_1$ is not a draw from a Gaussian distribution. Etc.

However, the dependence on $\boldsymbol{\Omega}_1$ is very weak.

Also, observe that any particular block of the test matrix gets "hit" by only L transforms, where L is the number of levels in the tree.

Empirically, the method works very well in practice, as long as the matrices $\mathbf{H}_j$ are well-conditioned with singular values close to 1.

Fix: "Refresh" the test matrix occasionally by redrawing a "pure" Gaussian matrix.

Randomized simultaneous factorization and compression (RSFC)

Vulnerability: Recall that RSFC requires repeated updating of the test matrix

$$\boldsymbol{\Omega} \quad \mapsto \quad \boldsymbol{\Omega}_1 = \mathbf{H}_1\boldsymbol{\Omega} \quad \mapsto \quad \boldsymbol{\Omega}_2 = \mathbf{H}_2\boldsymbol{\Omega}_1 \quad \mapsto \quad \boldsymbol{\Omega}_3 = \mathbf{H}_3\boldsymbol{\Omega}_2 \cdots$$

An issue arises in that $\mathbf{H}_1$ depends on $\boldsymbol{\Omega}$, so $\boldsymbol{\Omega}_1$ is not a draw from a Gaussian distribution. Etc.

However, the dependence on $\boldsymbol{\Omega}_1$ is very weak.

Also, observe that any particular block of the test matrix gets "hit" by only L transforms, where L is the number of levels in the tree.

Empirically, the method works very well in practice, as long as the matrices $\mathbf{H}_j$ are well-conditioned with singular values close to 1.

Fix: "Refresh" the test matrix occasionally by redrawing a "pure" Gaussian matrix.

Next: A version of RSFC that involves *only unitary transforms!*

Randomized simultaneous fact. and comp. (RSFC) – strong admissibility

Let us next consider the case of *strong* admissibility.

We will describe a version of RSFC that is based on the *strong recursive skeletonization (SRS)* algorithm of V. Minden, K. Ho, A. Damle, and L. Ying (Multiscale Modeling & Simulation, **15**(2), 2017).

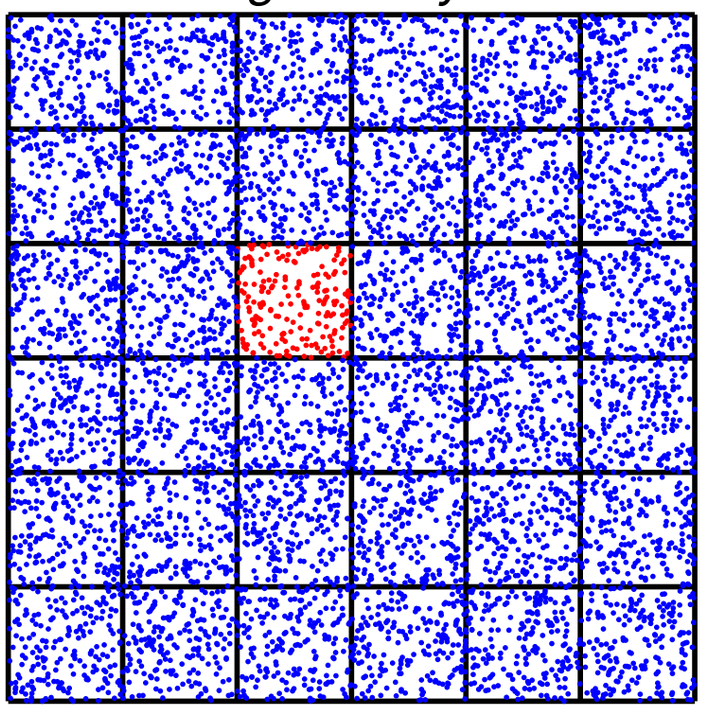
SRS has a compelling computational profile: A single pass through all nodes in a tree – going from smaller boxes to larger. *No nested hierarchies, no recursions.*

Importantly, all updates to the matrix are *multiplicative*. This makes SRS ideal for the "RSFC" environment.

Versions of rank structure: "strong" versus "weak" admissibility

Weak admissibility: Compress directly adjacent patches.

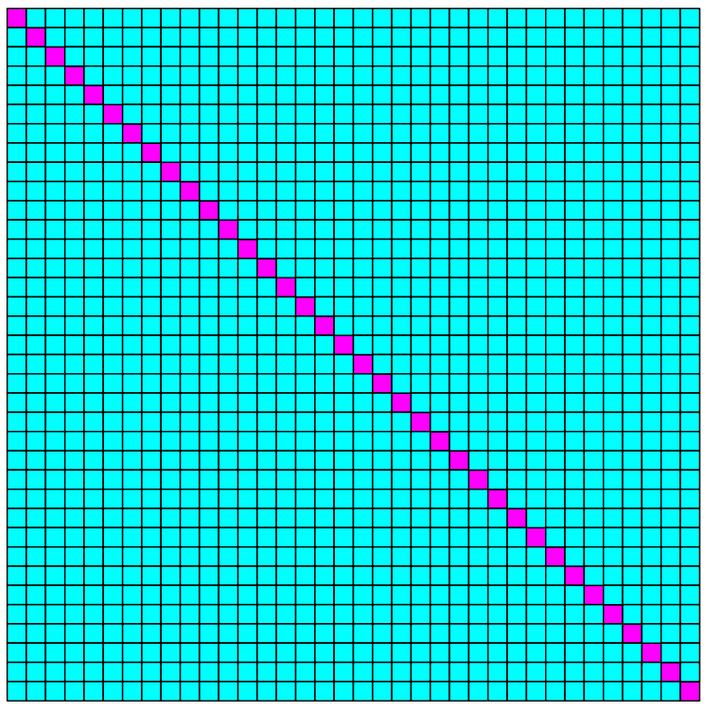
The geometry



Left: Points in a box $\Omega = [0,1]^2$. Red sources induce potentials on blue points. Average rank=13.9 at $\varepsilon = 10^{-8}$.

Right: Magenta blocks are dense. Cyan blocks low rank. Many low rank blocks, but high ranks.

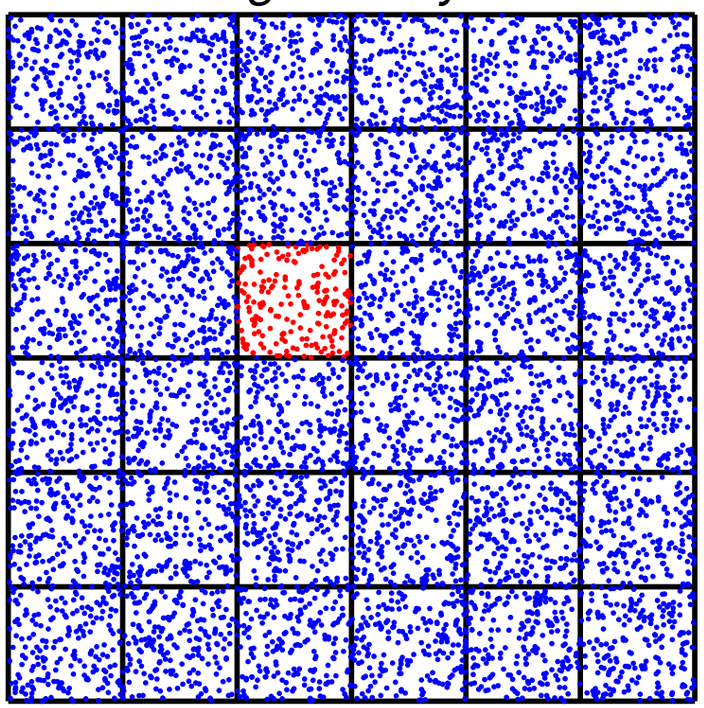
The matrix



Versions of rank structure: "strong" versus "weak" admissibility

Weak admissibility: Compress directly adjacent patches.

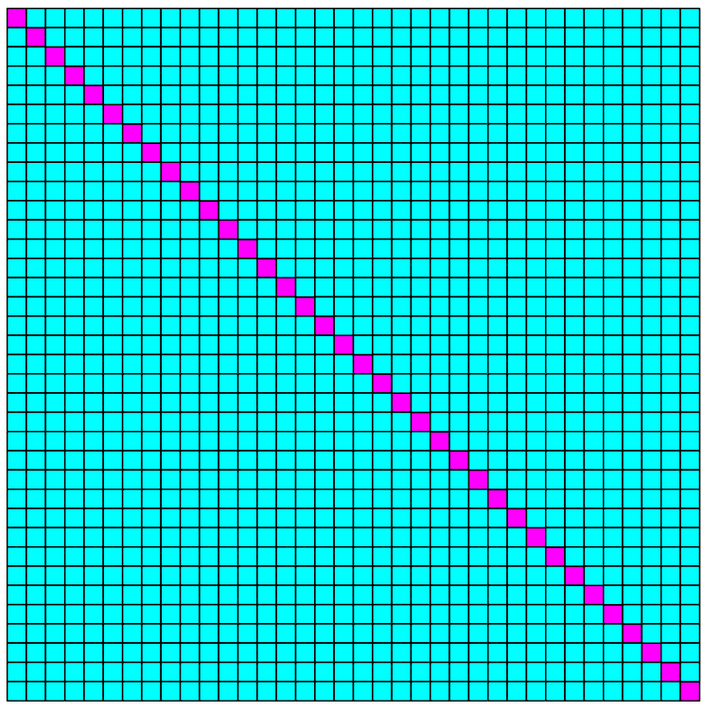
The geometry



Left: Points in a box $\Omega = [0,1]^2$. Red sources induce potentials on blue points. Average rank=13.9 at $\varepsilon = 10^{-8}$.

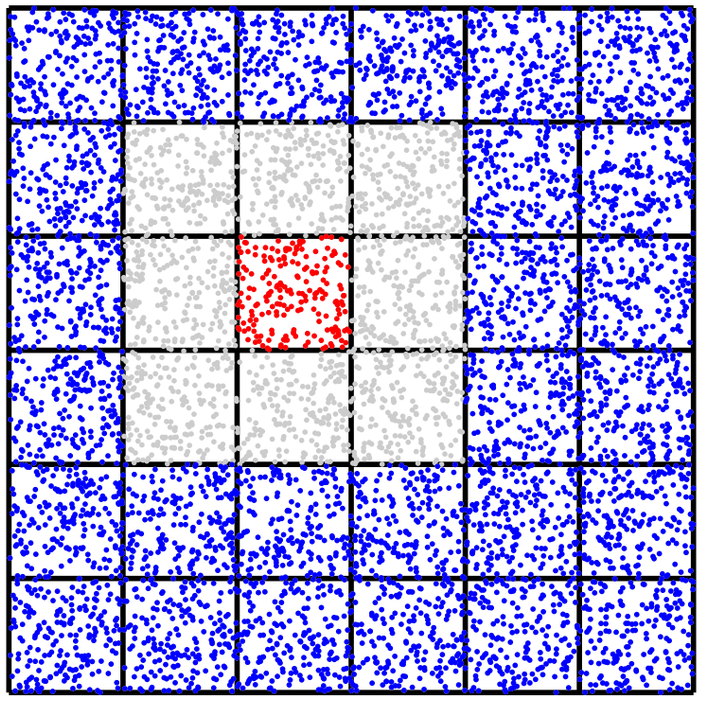
Right: Magenta blocks are dense. Cyan blocks low rank. Many low rank blocks, but high ranks.

The matrix



Strong admissibility: Compress only "far-field" interactions.

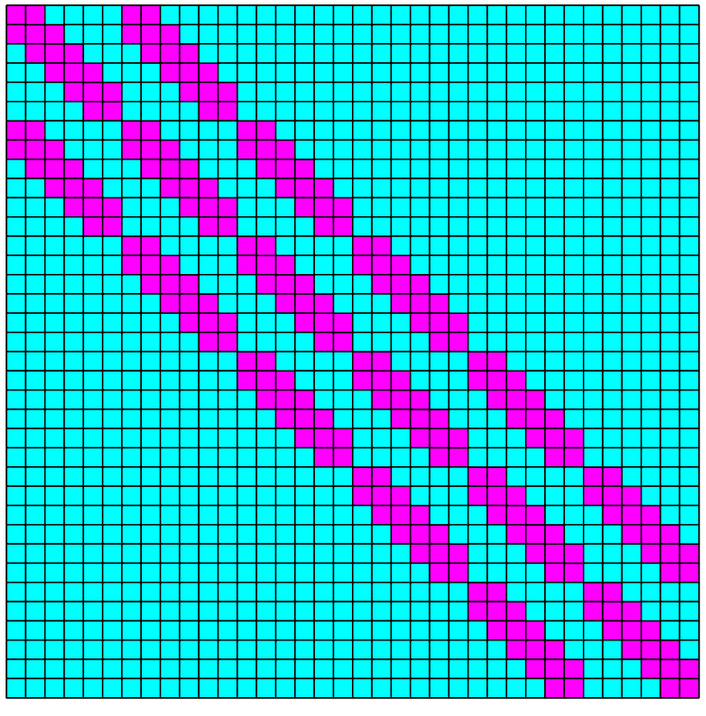
The geometry



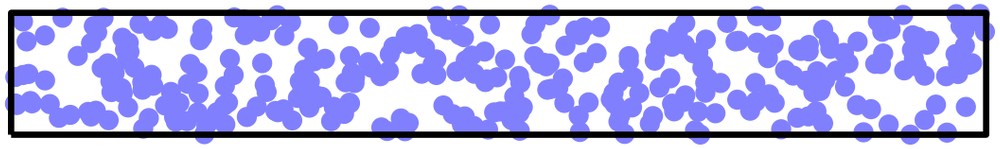
Left: Points in a box $\Omega = [0,1]^2$. Red sources induce potentials on blue points. Average rank=7.7 at $\varepsilon = 10^{-8}$.

Right: Magenta blocks are dense. Cyan blocks low rank. More dense blocks, but lower ranks.

The matrix



Model problem: Let $\{\boldsymbol{x}_i\}_{i=1}^N$ be a set of points in a rectangle Ω.



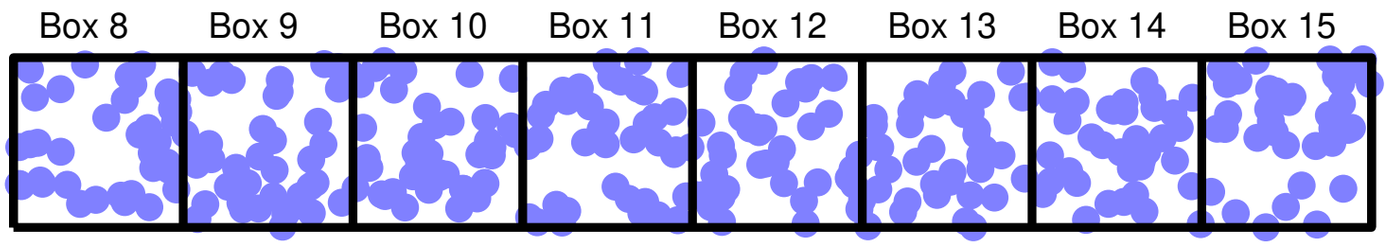
Let $\mathbf{A}$ be the $N \times N$ matrix with entries

$$\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \delta_{i,j}$$

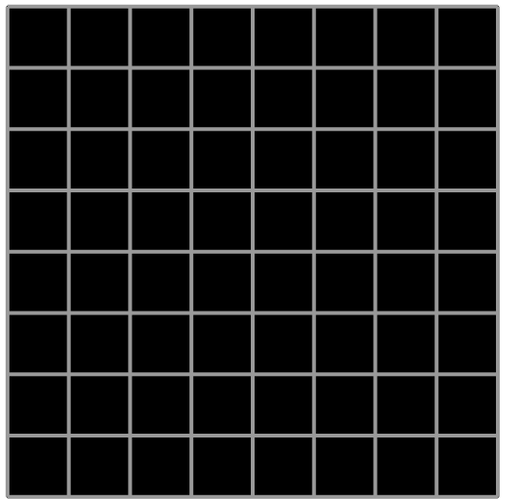
where k is some kernel function of mathematical physics such as $k(\boldsymbol{x}, \boldsymbol{y}) = \log|\boldsymbol{x} - \boldsymbol{y}|$.

We split the computational domain into eight blocks, which leads to a partition of $\mathbf{A}$.

Partitioning of the domain



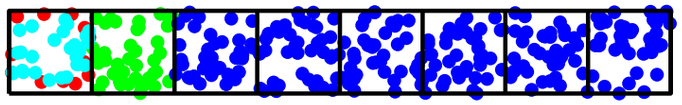
Tessellation of the matrix



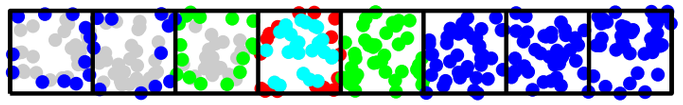
Our objective is now to drive $\mathbf{A}$ to block diagonal form through a sequence of inexpensive row and column operations.

Repeat the process with the remaining seven boxes:

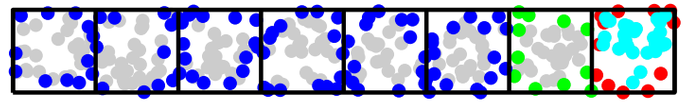
Geometry after step 1



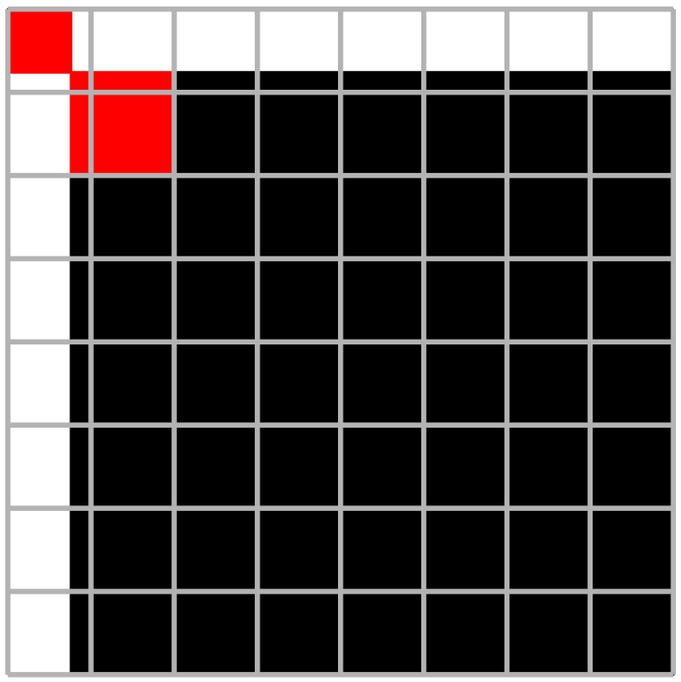
Geometry after step 4



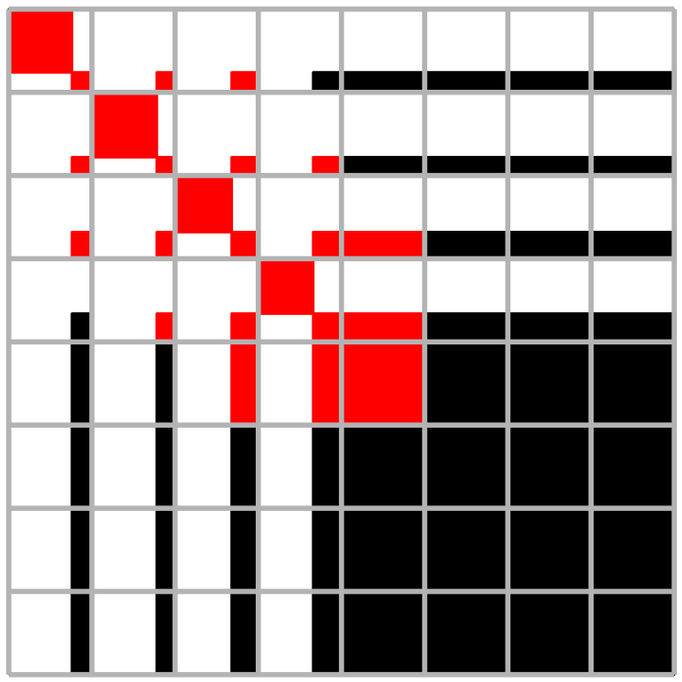
Geometry after step 8



Matrix after step 1



Matrix after step 4



Matrix after step 8

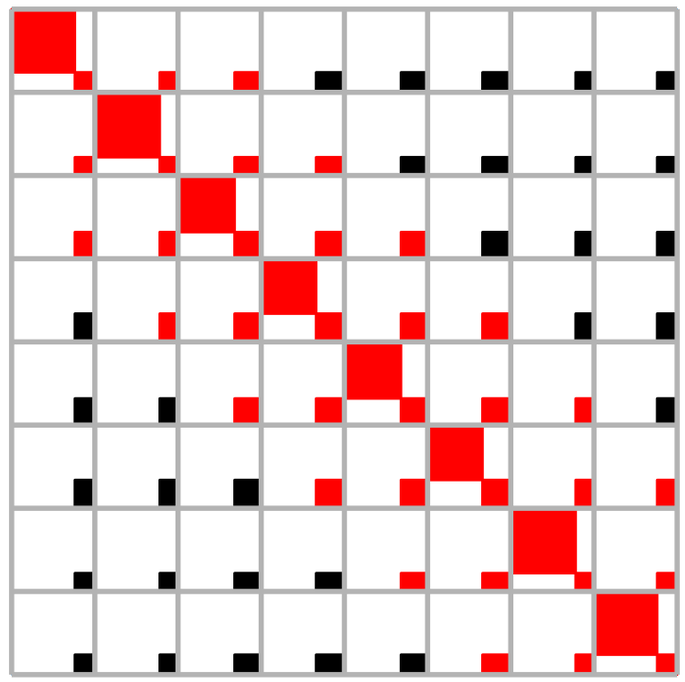
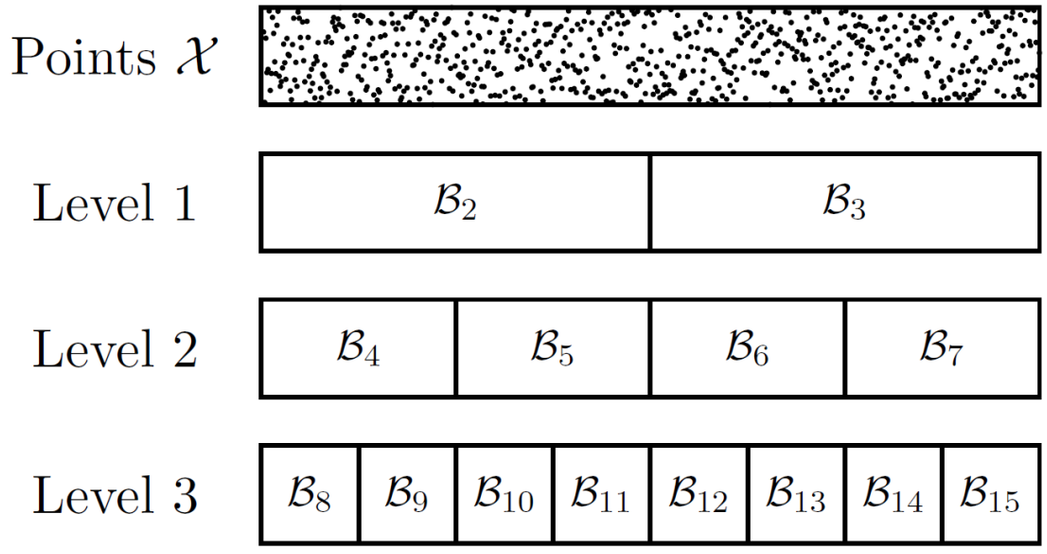


Illustration of the combined compression and factorization process. Upper row: The computational domain after steps 1, 4, and 8. The nodes are colored so that I_s is red, I_r is cyan, I_n is green, and I_f is blue. (The gray nodes mark points that were identified as "redundant" nodes in the previous steps. These points have dropped out of the computation and play no active role.) Lower row: The sparsity pattern of the matrix **A** *after the redundant nodes in Box 8/11/15 have been decoupled. White blocks are zeros, red blocks are entries that got modified, and black blocks are entries that have not been modified.*

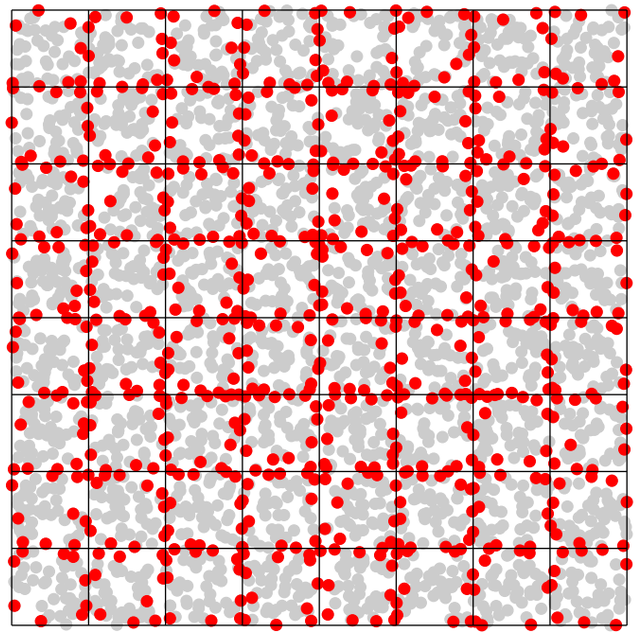
To get linear complexity, use a hierarchical tree:



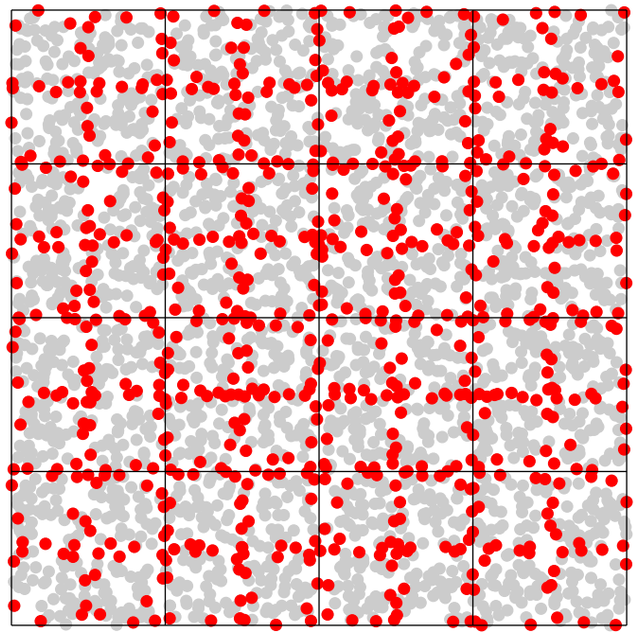
The hierarchical tessellation of boxes for the quasi one dimensional model problem.

A square domain

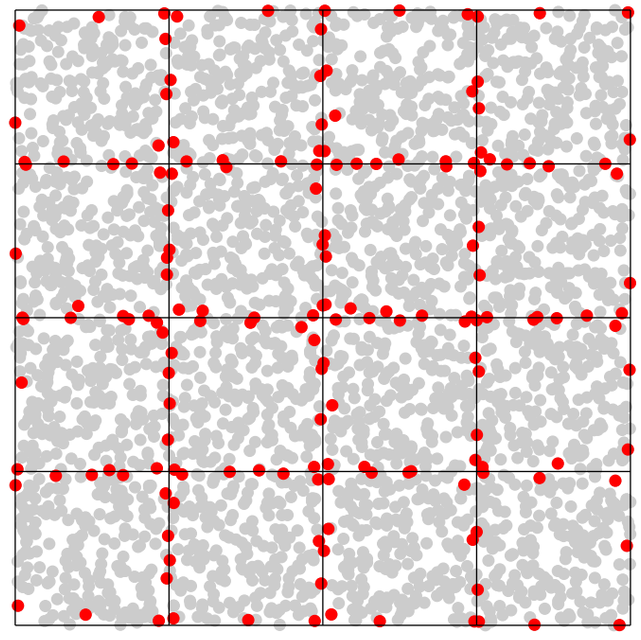
Active points after level 3:



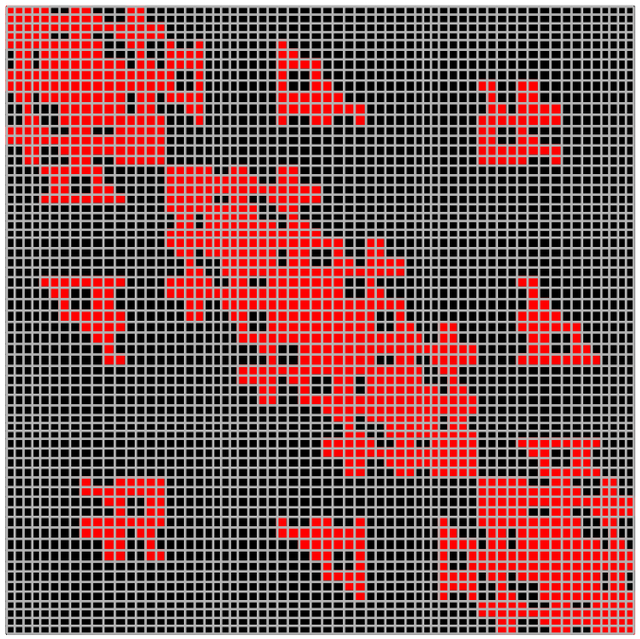
Merge boxes by fours:



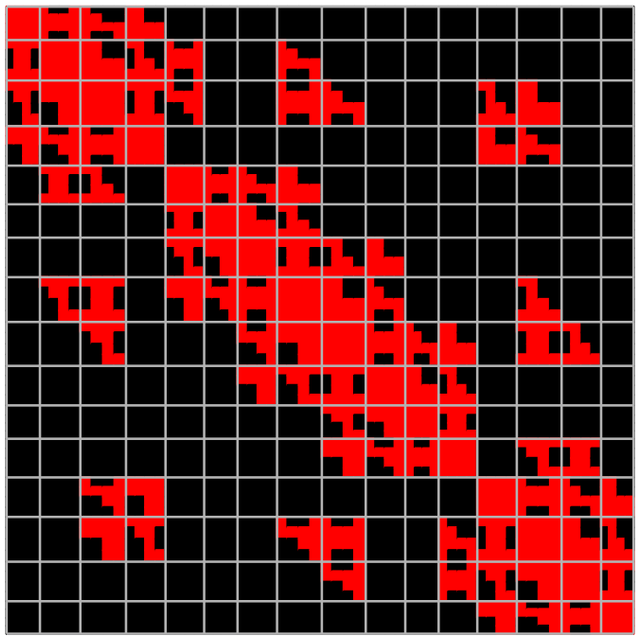
Active points after level 2:



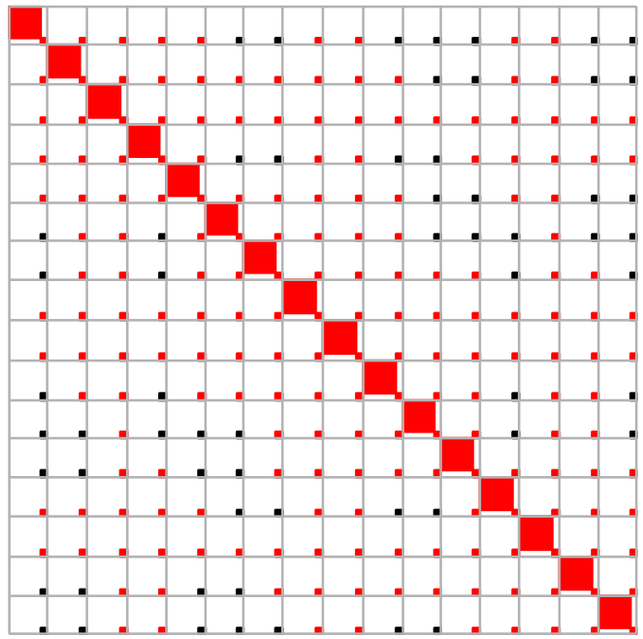
Matrix after level 3:



Matrix retessellated:



Matrix after level 2:



Observe that many more blocks get updated for a true two-dimensional domain — up to 25 boxes per row need to be explicitly stored.

Randomized strong recursive skeletonization (RSRS):

While strong recursive skeletonization (SRS) works well, the original version has significant limitations:

- Need to explicitly store *many* modified blocks of the matrix. (Possibly 125/row in 3D!)
- Complex dependencies between the different blocks. (So tricky to parallelize.)
- Reliance on "proxy surface" technique for compression limits range of applicability and can significantly overestimate the ranks.

Randomized strong recursive skeletonization (RSRS):

While strong recursive skeletonization (SRS) works well, the original version has significant limitations:

- Need to explicitly store *many* modified blocks of the matrix. (Possibly 125/row in 3D!)
- Complex dependencies between the different blocks. (So tricky to parallelize.)
- Reliance on "proxy surface" technique for compression limits range of applicability and can significantly overestimate the ranks.

New: RSRS is a randomized algorithm for *simultaneous compression and factorization:*

- Entirely black-box: Works whenever you have access to a fast technique for applying the matrix to a vector.
- No need to explicitly store updated off-diagonal blocks.
- Accurately detects the actual ranks of off-diagonal blocks.
- Linear overall complexity.
- Output is the SRS factorization of the matrix!
- Unitary transformations are now possible. $\rightarrow$ Better numerical stability.

A. Yesypenko, P.G. Martinsson, "Randomized Strong Recursive Skeletonization: Simultaneous compression and factorization of $\mathcal{H}^2$-matrices in the Black-Box Setting" arXiv:2311.01451, 2023. To appear in J. of Sci. Comp.

HPC implementation in progress – with T. Betcke, I. Fierro Picardo, and A. Yesypenko.

Numerical experiments on RSRS: Separate slides

Randomized numerical linear algebra

Primary benefits: reduced data movement, communication, and access requirements.

- Mature success stories include:
  - Low rank approximation. *Very high practical speed. Genuine theoretical breakthroughs.*
  - Over-determined least squares problems. *Rokhlin-Tygert, Sarlós, BLENDENPIK.*
  - Full rank revealing factorizations (CPQR, URV, ULV, ...). *Practical acceleration of $O(N^3)$ methods.*
- Current research directions include:
  - Extension to the case of tensors.
  - General linear system solvers: Partial success only. So far!
  - Variable precision computing: Randomization gives you a speed vs. accuracy knob.
    "Accurate enough" orthogonalization of Krylov vector. Approximate Hessians. Etc.
  - A posteriori error estimation: Go fast, estimate errors, correct as needed. "Responsibly reckless."
- 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 for building fast direct solvers for elliptic PDEs. *$O(N)$ sparse direct solvers.*
  - Path to solving oscillatory problems; and multi-physics and "multi-discretization" problems.

Links to survey articles on my webpage.