

The interpolatory decomposition and its applications in 21st century numerical analysis

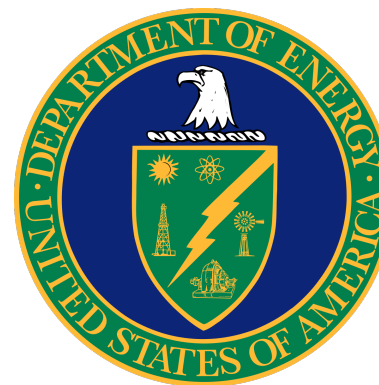
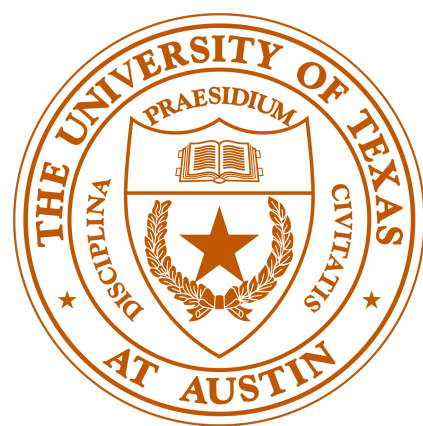
Gunnar Martinsson

Dept. of Mathematics & Oden Institute for Computational Sciences and Engineering
University of Texas at Austin

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

Current and recent students/postdocs: Yunhui Cai, Chao Chen, Yijun Dong, Nathan Heavner, James Levitt, Kate Pearce, Heather Wilber, Anna Yesyenko.

Research support by:



Outline of talk

- The interpolatory and CUR decompositions — what are they?
- Applications of IDs in solving PDEs and integral equations.
- Efficient algorithms for computing an interpolatory decomposition.
- PDE applications revisited: How to compress a global operator. *[Time permitting]*

Interpolative and CUR decompositions

Let \mathbf{A} be an $m \times n$ matrix of (approximate) rank k .

The *CUR* or *skeleton* approximation of \mathbf{A} takes the form

$$(1) \quad \begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{C} & \mathbf{U} & \mathbf{R}, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where \mathbf{C} holds a subset of the columns of \mathbf{A} , and \mathbf{R} holds a subset of the rows of \mathbf{A} .

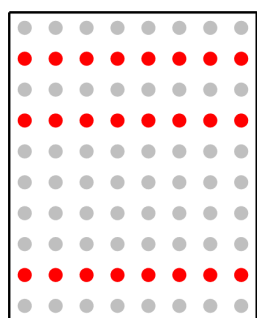
In other words, $\mathbf{C} = \mathbf{A}(:, J_S)$ and $\mathbf{R} = \mathbf{A}(I_S, :)$ for some index vectors J_S and I_S .

Closely related are the *interpolatory decompositions*

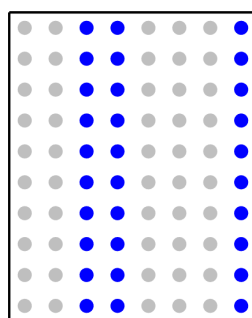
$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{C} & \mathbf{Z} & & \mathbf{A} & \approx & \mathbf{X} & \mathbf{R} & & \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}_S & \mathbf{Z} \\ m \times n & & m \times k & k \times n & & m \times n & & m \times k & k \times n & & m \times n & & m \times k & k \times k & k \times n \end{array}$$

where \mathbf{C} and \mathbf{R} are as in (1), and where $\mathbf{A}_S = \mathbf{A}(I_S, J_S)$.

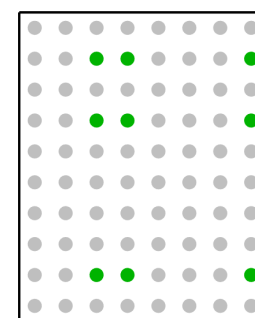
Example: Suppose \mathbf{A} is of size 10×8 and that $I_S = [2, 4, 9]$ and $J_S = [3, 4, 8]$. Then



$$\mathbf{R} = \mathbf{A}(I_S, :)$$



$$\mathbf{C} = \mathbf{A}(:, J_S)$$



$$\mathbf{A}_S = \mathbf{A}(I_S, J_S)$$

The objective is to find *spanning sets* of rows and columns.

Interpolative and CUR decompositions

Let \mathbf{A} be an $m \times n$ matrix of (approximate) rank k .

The *CUR* or *skeleton* approximation of \mathbf{A} takes the form

$$(1) \quad \begin{array}{ccccccc} \mathbf{A} & \approx & \mathbf{C} & \mathbf{U} & \mathbf{R}, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where \mathbf{C} holds a subset of the columns of \mathbf{A} , and \mathbf{R} holds a subset of the rows of \mathbf{A} .

In other words, $\mathbf{C} = \mathbf{A}(:, J_s)$ and $\mathbf{R} = \mathbf{A}(I_s, :)$ for some index vectors J_s and I_s .

Closely related are the *interpolatory decompositions*

$$\begin{array}{ccccccc} \mathbf{A} & \approx & \mathbf{C} & \mathbf{Z} & & & \\ m \times n & & m \times k & k \times n & & & \end{array} \quad \begin{array}{ccccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{R} & & & \\ m \times n & & m \times k & k \times n & & & \end{array} \quad \begin{array}{ccccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}_s & \mathbf{Z} & & \\ m \times n & & m \times k & k \times k & k \times n & & \end{array}$$

where \mathbf{C} and \mathbf{R} are as in (1), and where $\mathbf{A}_s = \mathbf{A}(I_s, J_s)$.

The fact that the matrices \mathbf{R} , \mathbf{C} , and \mathbf{A}_s are submatrices of \mathbf{A} has important advantages:

- If \mathbf{A} is sparse, then the factors \mathbf{R} and \mathbf{C} are sparse.
- If \mathbf{A} is non-negative, then the factors \mathbf{R} , \mathbf{C} , and \mathbf{A}_s are non-negative.
- The factorizations allow for data interpretation.
- Reduced storage, since we can store the index vectors rather than the factors.
- *Invaluable in the context of modern Fast Multipole Methods and Fast Direct Solvers.*

Special case: Exact rank deficiency. Suppose \mathbf{A} is of size $m \times n$ and *exact* rank k .

Let I and J be permutations of the row and column indices, and split \mathbf{A} into four parts

$$\mathbf{A}(I, J) = \begin{bmatrix} \mathbf{A}_{SS} & \mathbf{A}_{SR} \\ \mathbf{A}_{RS} & \mathbf{A}_{RR} \end{bmatrix},$$

so that \mathbf{A}_{SS} is the leading $k \times k$ submatrix. If \mathbf{A}_{SS} is non-singular, then necessarily

$$(2) \quad \mathbf{A}_{RR} = \mathbf{A}_{RS} \mathbf{A}_{SS}^{-1} \mathbf{A}_{SR}.$$

The *skeleton factorization* (CUR) follows directly from (2):

$$\mathbf{A}(I, J) = \underbrace{\begin{bmatrix} \mathbf{A}_{SS} \\ \mathbf{A}_{RS} \end{bmatrix}}_{=\mathbf{C}} \underbrace{\mathbf{A}_{SS}^{-1}}_{=\mathbf{U}} \underbrace{\begin{bmatrix} \mathbf{A}_{SS} & \mathbf{A}_{SR} \end{bmatrix}}_{=\mathbf{R}}.$$

Analogously, the *interpolatory decomposition* takes the form

$$\mathbf{A}(I, J) = \underbrace{\begin{bmatrix} \mathbf{I} \\ \mathbf{A}_{RS} \mathbf{A}_{SS}^{-1} \end{bmatrix}}_{=\mathbf{X}} \underbrace{\mathbf{A}_{SS}}_{=\mathbf{A}_S} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{A}_{SS}^{-1} \mathbf{A}_{SR} \end{bmatrix}}_{=\mathbf{Z}}.$$

So existence of the CUR and the ID are straight-forward. But two questions arise:

(1) Are the factorizations well-conditioned?

(2) For a matrix of only *approximate* rank k , how close to optimal can you get?

Conditioning of the interpolatory decomposition

Recall the interpolatory decomposition:

$$\begin{array}{ccccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}_S & \mathbf{Z} & & \\ m \times n & & m \times k & k \times k & k \times n & & \end{array}$$

where $\mathbf{A}_S = \mathbf{A}(I_S, J_S)$ is a $k \times k$ submatrix of \mathbf{A} .

\mathbf{X} and \mathbf{Z} hold $k \times k$ identity matrices as submatrices. $\Rightarrow \sigma_{\min}(\mathbf{X}) = \sigma_{\min}(\mathbf{Z}) = 1$

So \mathbf{X} and \mathbf{Z} are well-conditioned iff their *maximal* singular values are controlled.

Claim: Pick I_S and J_S so that $|\det(\mathbf{A}(I_S, J_S))|$ is the max over all $k \times k$ submatrices. Then

$$(3) \quad \sup_{i,j} |\mathbf{X}(i,j)| \leq 1 \quad \text{and} \quad \sup_{i,j} |\mathbf{Z}(i,j)| \leq 1.$$

Proof: Use Cramer's rule to bound entries of $\mathbf{A}_{rs} \mathbf{A}_{ss}^{-1}$ and $\mathbf{A}_{ss}^{-1} \mathbf{A}_{sr}$. □

The bounds (3) imply that $\kappa(\mathbf{X}) \leq \sqrt{1 + k(m - k)}$ and $\kappa(\mathbf{Z}) \leq \sqrt{1 + k(n - k)}$.

So **YES**, a reasonably well-conditioned ID exists. (Finding it is another matter ...)

Note: CUR is in general *not* well conditioned.

Optimality in terms of low rank approximation

Recall Eckart–Young theorem:

$$\inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\} = \sigma_{k+1}(\mathbf{A})$$

(Our default is that $\|\cdot\|$ refers to the ℓ^2 operator norm.)

Question: How close to optimal can you get with an ID or a CUR decomposition?

Optimality in terms of low rank approximation

Recall Eckart–Young theorem: $\inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\} = \sigma_{k+1}(\mathbf{A})$

(Our default is that $\|\cdot\|$ refers to the ℓ^2 operator norm.)

Question: How close to optimal can you get with an ID or a CUR decomposition?

Very well studied subject.

Tends to be within factor $\sim \sqrt{kn}$ of optimal in the worst case.

Often much better in practice, in particular when the singular values decay rapidly.

Theorem: Let \mathbf{A} be an $m \times n$ matrix, and let $k < \min(m, n)$. There exists a matrix \mathbf{U} of size $k \times k$, and index vectors I_S and J_S of length k such that

$$\|\mathbf{A} - \mathbf{CUR}\| \leq (1 + 2\sqrt{k}(\sqrt{m} + \sqrt{n}))\sigma_{k+1}(\mathbf{A}),$$

where

$$\mathbf{C} = \mathbf{A}(:, J_S), \quad \mathbf{R} = \mathbf{A}(I_S, :).$$

References: Goreinov, S.A., Tyrtyshnikov, E.E., Zamarashkin, N.L. (1997); Goreinov, S.A., Tyrtyshnikov, E.E. (2001). Survey: Ballani, J. & Kressner, D. (2016).

Example: Finding sparse solutions to least squares problems

Given a rank deficient matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{R}^{m \times 1}$, consider the problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{Ax} - \mathbf{b}\|.$$

The *minimal norm* solution can be computed via the SVD.

A *sparse solution* can be computed via an ID (or, in practice, a rank revealing QR).

If $\mathbf{A} = \mathbf{CZ}$, then

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{Ax} - \mathbf{b}\| = \min_{\mathbf{x}' \in \mathbb{R}^k} \|\mathbf{Cx}' - \mathbf{b}\|.$$

If the ID is well-conditioned, the norm of the solution can be controlled.

Very simple idea, with surprisingly interesting applications.

Example: Finding sparse solutions to least squares problems

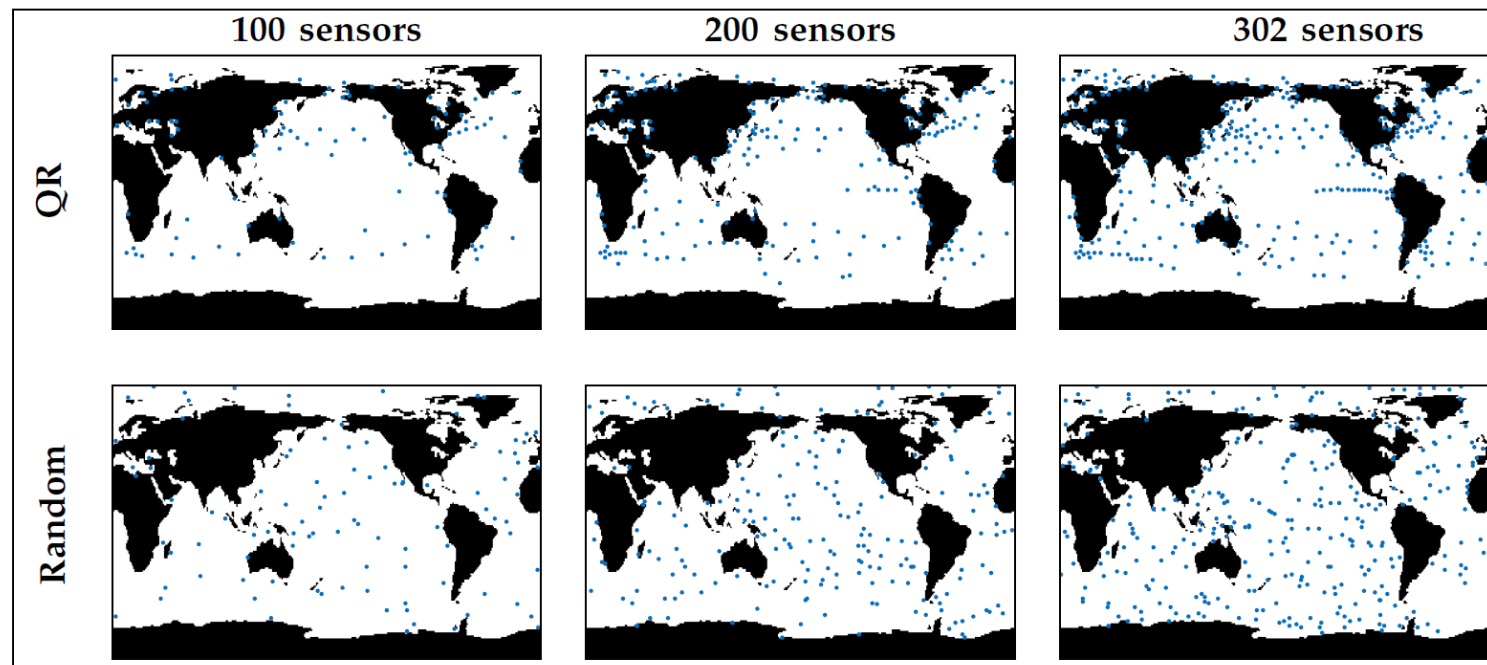
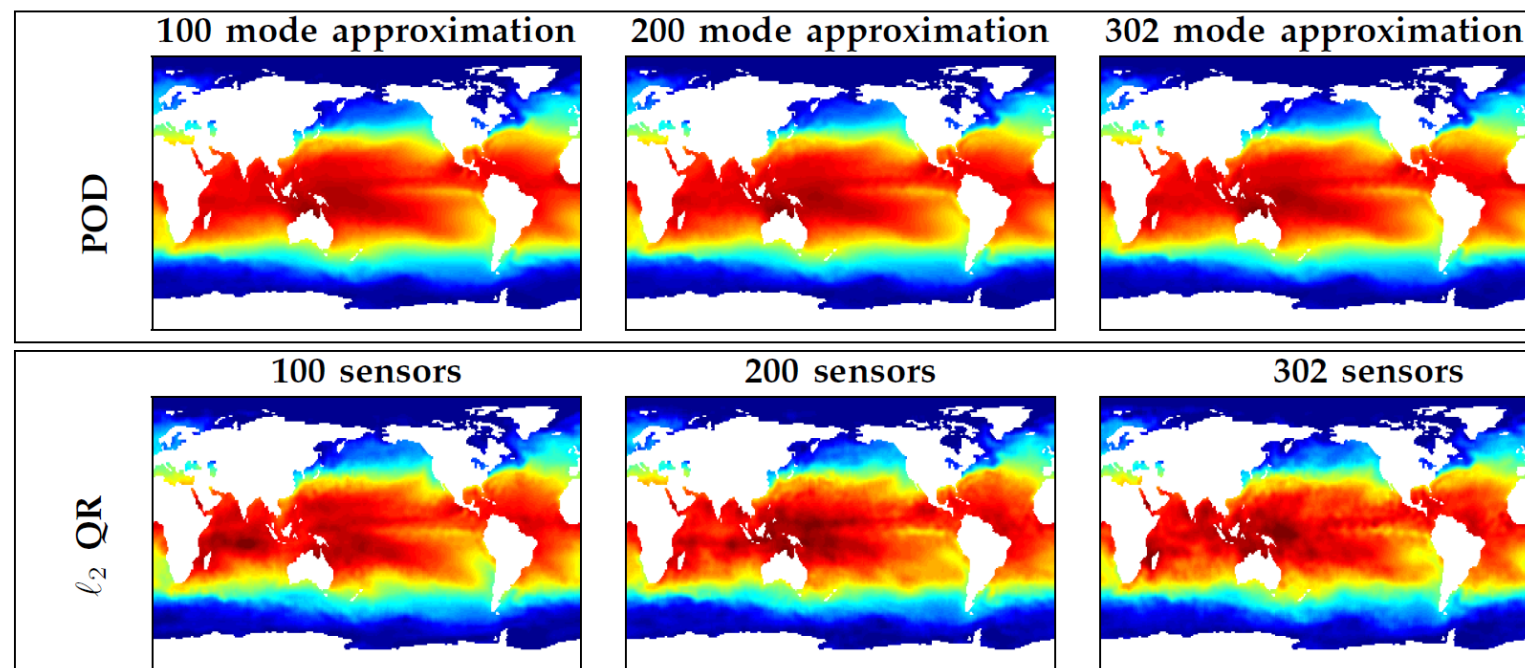


Figure 10: QR selected sensors used for reconstruction. QR sensors are informative about ocean dynamics, for example capturing convective phenomena such as the El Niño Southern Oscillation off coastal Peru.



*From “Data-Driven Sparse Sensor Placement for Reconstruction”
by Manohar, Brunton, Kutz, & Brunton*

Example: Method of fundamental solutions (special case of least squares problem)

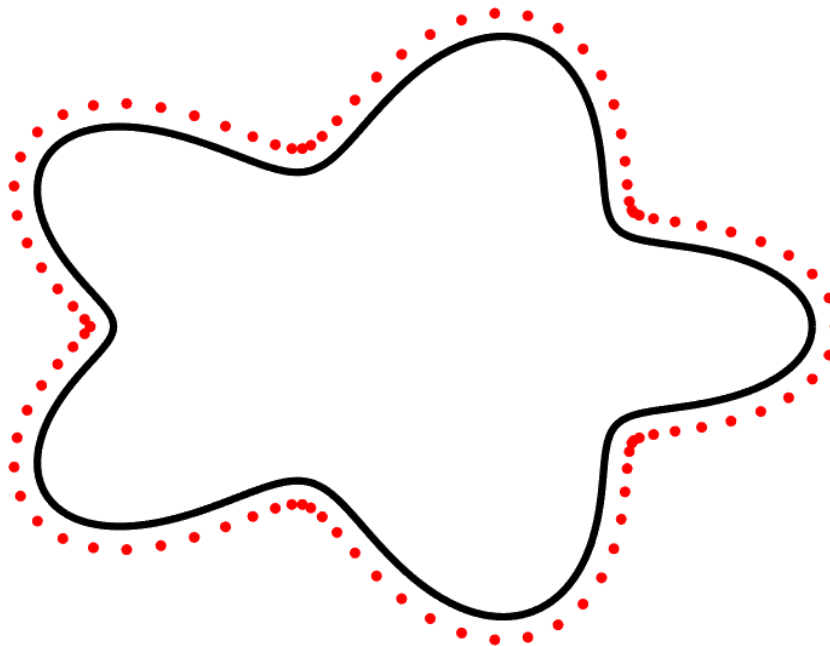
Consider a Dirichlet boundary value problem on a domain Ω with boundary Γ :

$$\begin{cases} -\Delta u(\mathbf{x}) = 0, & \mathbf{x} \in \Omega \\ u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma \end{cases}$$

We seek a solution of the form

$$u(\mathbf{x}) = \sum_{j=1}^n \log |\mathbf{x} - \mathbf{y}_j| q_j$$

where $\{\mathbf{y}_j\}_{j=1}^n$ denotes a set of source locations outside of Ω .



The red dots denote the points \mathbf{y}_j .

Example: Method of fundamental solutions (special case of least squares problem)

Consider a Dirichlet boundary value problem on a domain Ω with boundary Γ :

$$\begin{cases} -\Delta u(\mathbf{x}) = 0, & \mathbf{x} \in \Omega \\ u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma \end{cases}$$

We seek a solution of the form

$$u(\mathbf{x}) = \sum_{j=1}^n \log |\mathbf{x} - \mathbf{y}_j| q_j$$

where $\{\mathbf{y}_j\}_{j=1}^n$ denotes a set of source locations outside of Ω .

Specifying a set of collocation points $\{\mathbf{x}_i\}_{i=1}^m \subset \Gamma$, our task is then to solve the system

$$\sum_{j=1}^n \log |\mathbf{x}_i - \mathbf{y}_j| q_j = f(\mathbf{x}_i), \quad i \in \{1, 2, \dots, m\}$$

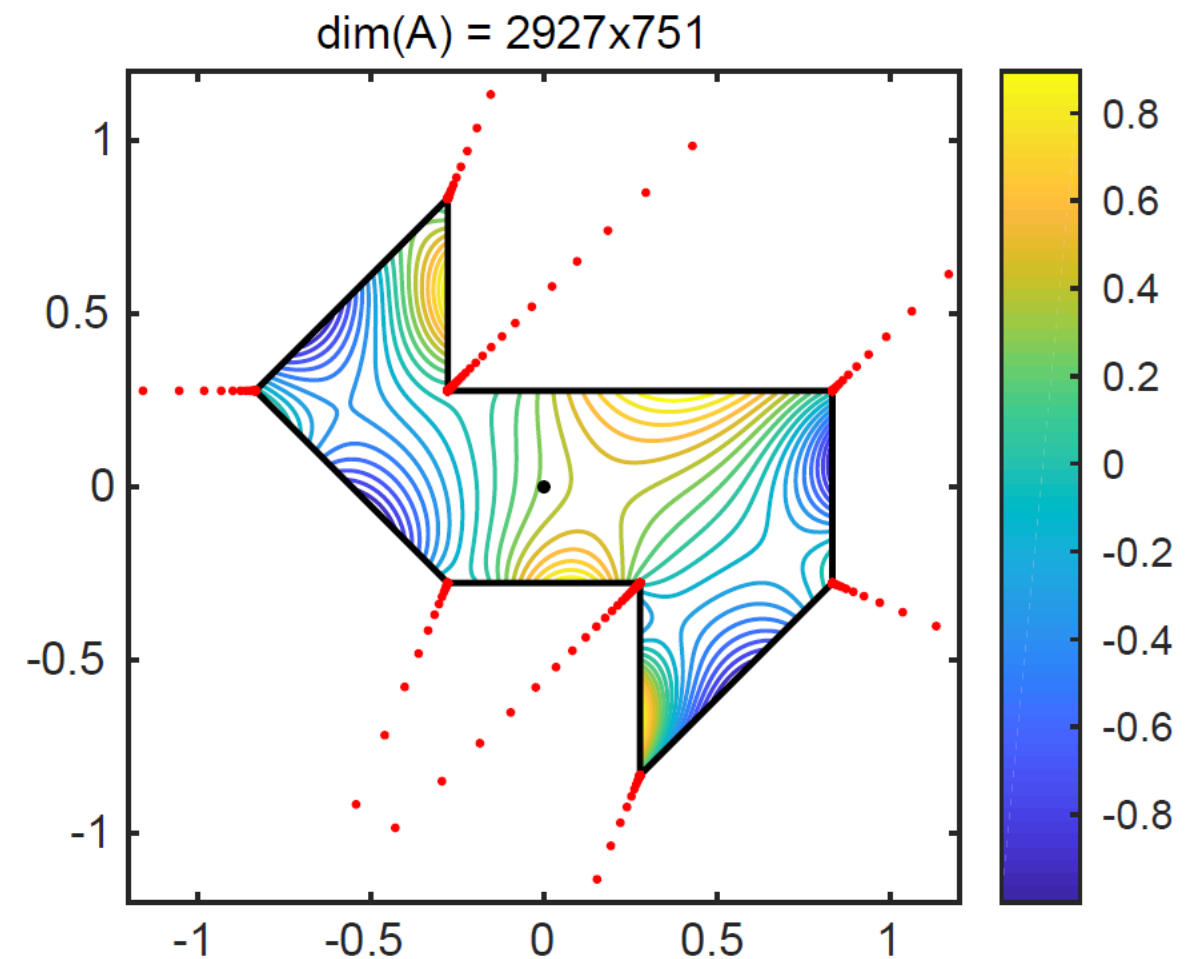
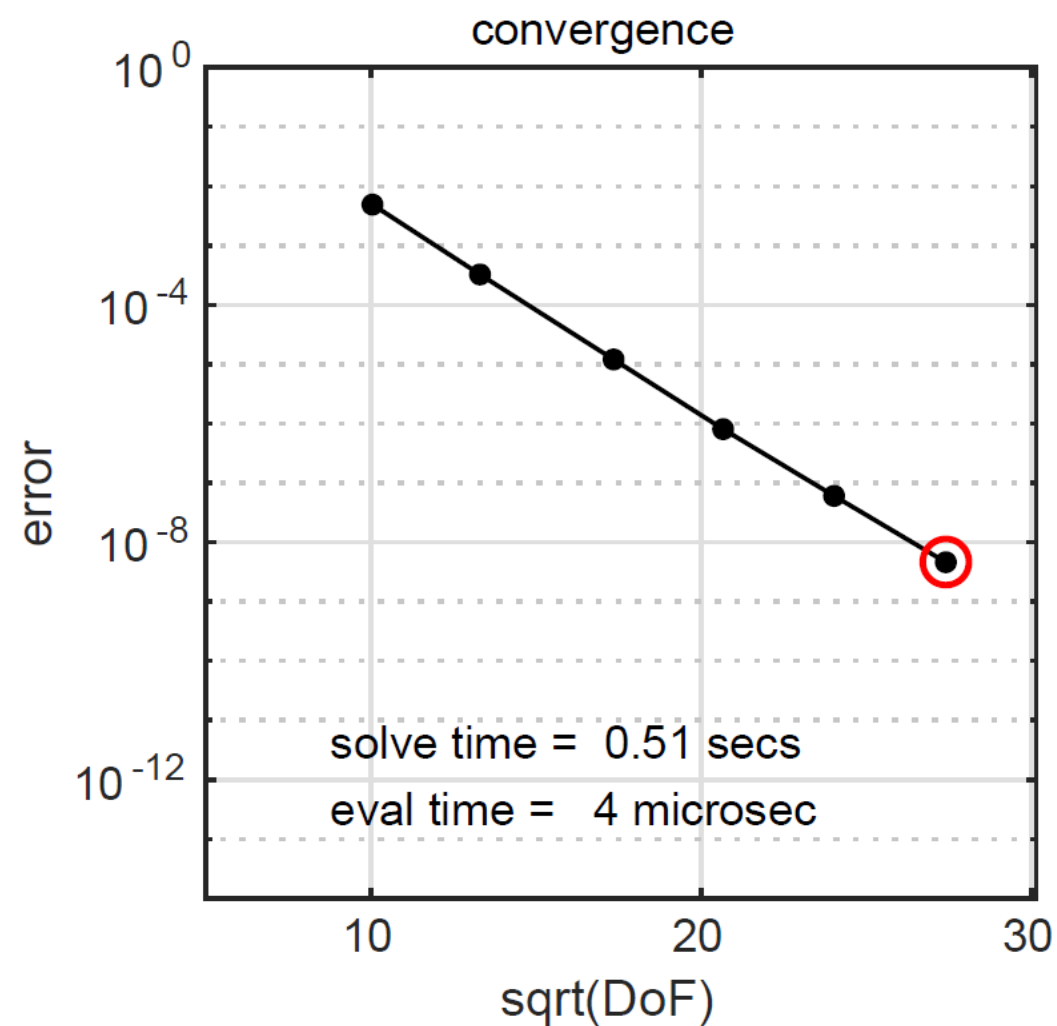
in a least squares sense.

The matrix \mathbf{A} with entries $\mathbf{A}(i, j) = \log |\mathbf{x}_i - \mathbf{y}_j|$ is typically numerically rank deficient, and an ID can help reduce the dimensionality of the problem.

Rectangular world! Cf. talk by D. Huybrechs; Barnett & Betcke 2007; ...

Example: Method of fundamental solutions (special case of least squares problem)

With analysis and extra bells and whistles, domains with corners can be handled too!



From Gopal & Trefethen, SINUM, 57(4), 2019 – “Lightning solver”

Similar problems arise in some variations of “radial basis function” methods.

Backwards stability is crucial in this context.

Example: Recursive skeletonization (a fast direct solver for integral equations)

Consider an integral equation

$$(BIE) \quad q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma,$$

where Γ is a contour in \mathbb{R}^2 or a surface in \mathbb{R}^3 .

Example: Equation (BIE) may be a reformulation of Laplace BVP

$$\begin{cases} -\Delta u(\mathbf{x}) = 0, & \mathbf{x} \in \Omega \\ u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma \end{cases}$$

The standard BIE formulation here is

$$\frac{1}{2}q(\mathbf{x}) + \int_{\Gamma} d(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma,$$

where

$$d(\mathbf{x}, \mathbf{y}) = \mathbf{n}(\mathbf{y}) \cdot \nabla_{\mathbf{y}} \frac{\log |\mathbf{x} - \mathbf{y}|}{2\pi} = \frac{\mathbf{n}(\mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})}{2\pi |\mathbf{x} - \mathbf{y}|^2}.$$

Cf. talks by Anna-Karin Tornberg, Abi Gopal, Hadrien Montanelli, et al.

Example: Recursive skeletonization (a fast direct solver for integral equations)

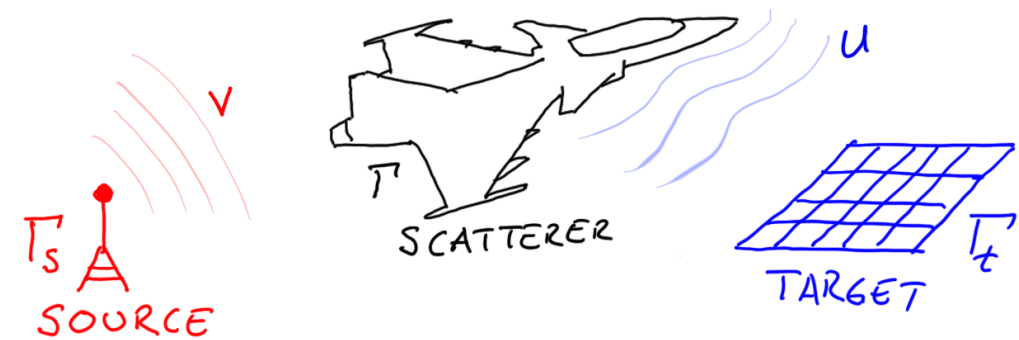
Consider an integral equation

$$(BIE) \quad q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma,$$

where Γ is a contour in \mathbb{R}^2 or a surface in \mathbb{R}^3 .

Example:

Consider the problem of (sound-soft) acoustic scattering from a finite body. The governing equations may be:



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

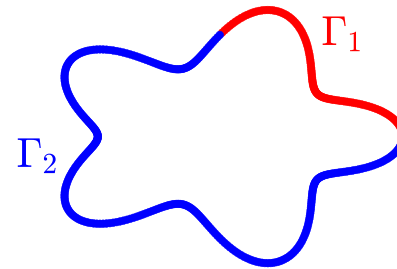
The PDE (BVP) has an alternative mathematical formulation in the BIE

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

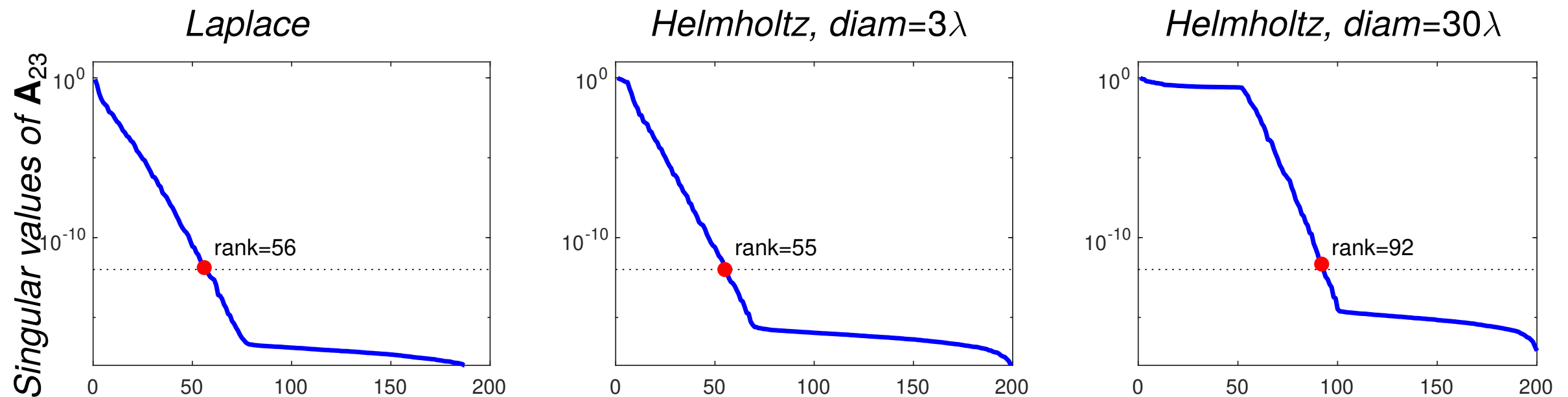
Example: Recursive skeletonization (a fast direct solver for integral equations)

Let \mathbf{A} be a dense matrix arising from discretizing an integral equation, and partition

A_{11}	A_{12}
A_{21}	A_{22}



Claim: The matrices \mathbf{A}_{12} and \mathbf{A}_{21} often have low numerical rank. Typical behavior:

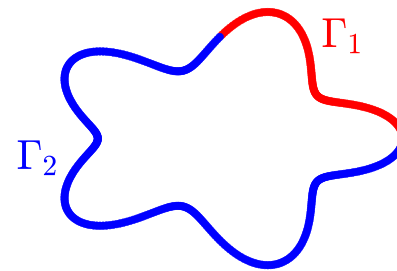


Important: 3D is more adversarial. Slightly different representations are called for.

Example: Recursive skeletonization (a fast direct solver for integral equations)

Let \mathbf{A} be a dense matrix arising from discretizing an integral equation, and partition

$$\begin{array}{|c|c|} \hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline \end{array}$$



To reveal the numerical rank, we form the IDs of \mathbf{A}_{12} and \mathbf{A}_{21} to obtain

$$\mathbf{A}_{12} = \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \quad \text{and} \quad \mathbf{A}_{21}^* = \mathbf{W}_1 \tilde{\mathbf{A}}_{21}^*$$

The matrix \mathbf{U}_1 takes the form

$$\mathbf{U}_1 = \mathbf{P}_1 \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{T} & \mathbf{I} \end{bmatrix},$$

where \mathbf{P}_1 is a permutation matrix, and where \mathbf{T} is a matrix that holds the expansion coefficients in the interpolatory decomposition. \mathbf{U}_1 is not unitary, but it is invertible, with

$$\mathbf{U}_1^{-1} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{T} & \mathbf{I} \end{bmatrix} \mathbf{P}_1^*,$$

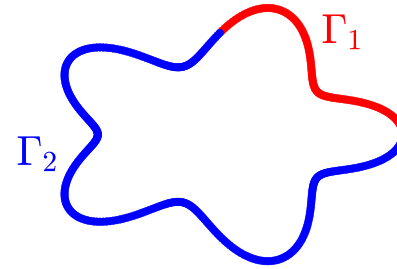
(Recall that there exists an ID s.t. $|\mathbf{T}(i,j)| \leq 1 \quad \forall i,j$, so the conditioning is at least ok.)

Note: For notational simplicity, we assume *exact* rank deficiencies in this discussion.

Example: Recursive skeletonization (a fast direct solver for integral equations)

Let \mathbf{A} be a dense matrix arising from discretizing an integral equation, and partition

A_{11}	A_{12}
A_{21}	A_{22}



To reveal the numerical rank, we form the IDs of \mathbf{A}_{12} and \mathbf{A}_{21} to obtain

$$\mathbf{A}_{12} = \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \quad \text{and} \quad \mathbf{A}_{21}^* = \mathbf{W}_1 \tilde{\mathbf{A}}_{21}^*$$

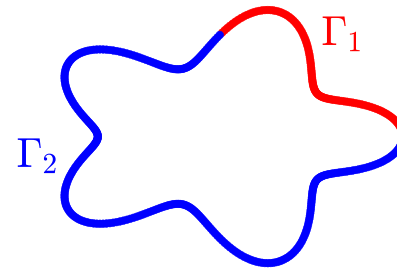
The diagram shows the matrix factorizations with visual representations of the matrices: \mathbf{A}_{12} is a large black rectangle, \mathbf{U}_1 is a small black square, and $\tilde{\mathbf{A}}_{12}$ is a wide, thin white rectangle. Similarly, \mathbf{A}_{21}^* is a large black rectangle, \mathbf{W}_1 is a small black square, and $\tilde{\mathbf{A}}_{21}^*$ is a wide, thin white rectangle.

Let us plug the factorizations into \mathbf{A} :

Example: Recursive skeletonization (a fast direct solver for integral equations)

Let \mathbf{A} be a dense matrix arising from discretizing an integral equation, and partition

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$



To reveal the numerical rank, we form the IDs of \mathbf{A}_{12} and \mathbf{A}_{21} to obtain

$$\mathbf{A}_{12} = \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \quad \text{and} \quad \mathbf{A}_{21}^* = \mathbf{W}_1 \tilde{\mathbf{A}}_{21}^*$$

Let us plug the factorizations into \mathbf{A} :

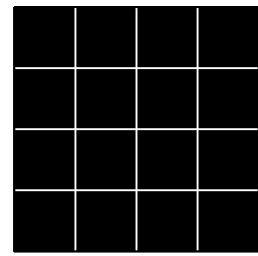
$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \\ \tilde{\mathbf{A}}_{21} \mathbf{W}_1^* & \mathbf{A}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{U}_1^{-1} \mathbf{A}_{11} \mathbf{W}_1^{-*} & \tilde{\mathbf{A}}_{12} \\ \tilde{\mathbf{A}}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{W}_1^* & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

*The matrices $\tilde{\mathbf{A}}_{12}$ and $\tilde{\mathbf{A}}_{21}$ do not need to be computed since the non-zero elements form **submatrices of \mathbf{A} .***

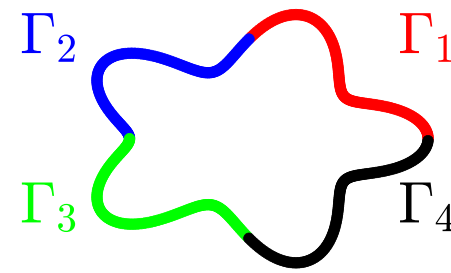
(It would have been possible to use QR or SVD to build the basis matrices \mathbf{U}_1 and \mathbf{W}_1 . But then you would mix all entries and would have to explicitly compute $\tilde{\mathbf{A}}_{12}$ and $\tilde{\mathbf{A}}_{21}$.)

Example: Recursive skeletonization (a fast direct solver for integral equations)

Let \mathbf{A} be a dense matrix arising from discretizing an integral equation, and partition



The matrix



Partitioning of Γ

Next let us consider a 4×4 partitioning of \mathbf{A} .

The ID now takes the form

$$\mathbf{A}_{ij} = \mathbf{U}_i \tilde{\mathbf{A}}_{ij} \mathbf{W}_j^* \quad i, j \in \{1, 2, 3, 4\}, \quad i \neq j.$$

This leads to the factorization

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} & \mathbf{A}_{24} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} & \mathbf{A}_{34} \\ \mathbf{A}_{41} & \mathbf{A}_{42} & \mathbf{A}_{43} & \mathbf{A}_{44} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_1 & 0 & 0 & 0 \\ 0 & \mathbf{U}_2 & 0 & 0 \\ 0 & 0 & \mathbf{U}_3 & 0 \\ 0 & 0 & 0 & \mathbf{U}_4 \end{bmatrix} \begin{bmatrix} \mathbf{U}_1^{-1} \mathbf{A}_{11} \mathbf{W}_1^{-*} & \tilde{\mathbf{A}}_{12} & \tilde{\mathbf{A}}_{13} & \tilde{\mathbf{A}}_{14} \\ \tilde{\mathbf{A}}_{21} & \mathbf{U}_2^{-1} \mathbf{A}_{22} \mathbf{W}_2^{-*} & \tilde{\mathbf{A}}_{23} & \tilde{\mathbf{A}}_{24} \\ \tilde{\mathbf{A}}_{31} & \tilde{\mathbf{A}}_{32} & \mathbf{U}_3^{-1} \mathbf{A}_{33} \mathbf{W}_3^{-*} & \tilde{\mathbf{A}}_{34} \\ \tilde{\mathbf{A}}_{41} & \tilde{\mathbf{A}}_{42} & \tilde{\mathbf{A}}_{43} & \mathbf{U}_4^{-1} \mathbf{A}_{44} \mathbf{W}_4^{-*} \end{bmatrix} \begin{bmatrix} \mathbf{W}_1^* & 0 & 0 & 0 \\ 0 & \mathbf{W}_2^* & 0 & 0 \\ 0 & 0 & \mathbf{W}_3^* & 0 \\ 0 & 0 & 0 & \mathbf{W}_4^* \end{bmatrix}$$

The non-zero elements of $\tilde{\mathbf{A}}_{ij}$ form **submatrices of \mathbf{A}** .

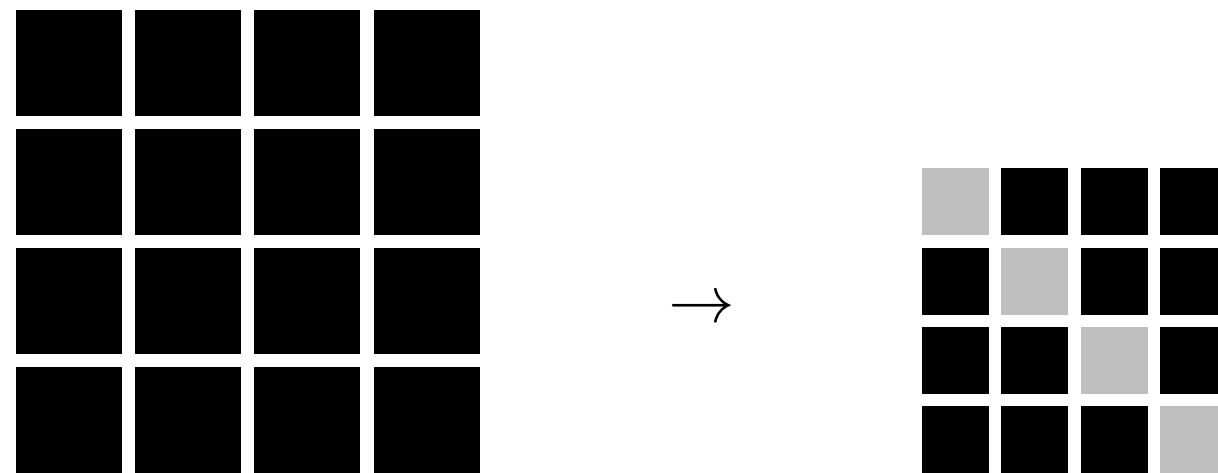
Example: Recursive skeletonization (a fast direct solver for integral equations)

We have built a scheme for reducing a system of size $pn \times pn$ to one of size $pk \times pk$, where

p is the number of diagonal blocks,

n is the size of a diagonal block,

k is the rank, which is the size of the reduced blocks.



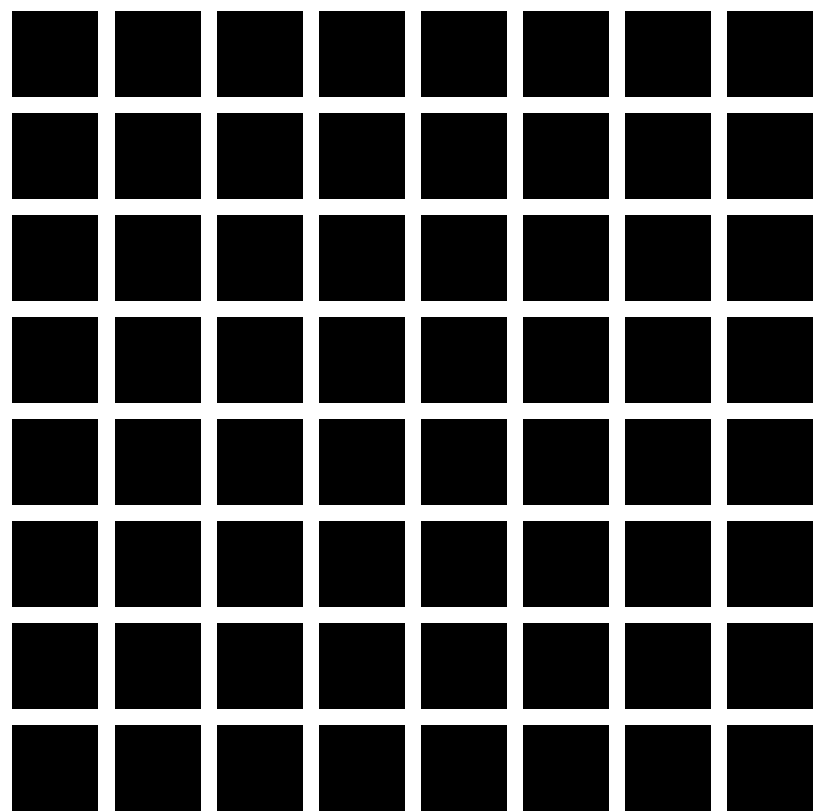
The computational gain is $(k/n)^3$. Good, but not earth-shattering.

Question: How do we get to $O(N)$?

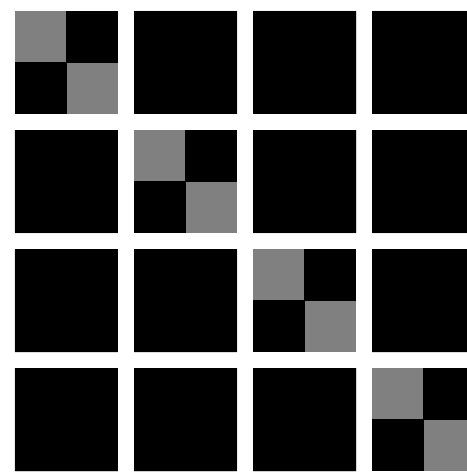
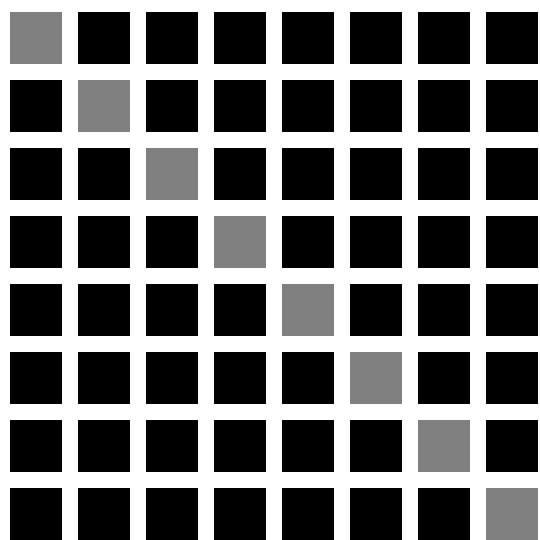
Answer: It turns out that the reduced matrix is itself compressible. Recurse!

Example: Recursive skeletonization (a fast direct solver for integral equations)

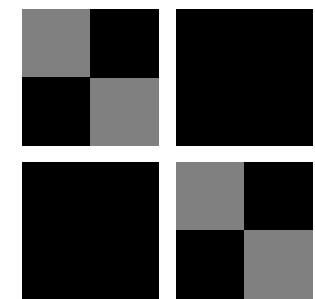
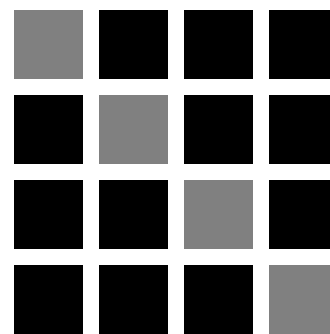
A globally $O(N)$ algorithm is obtained by hierarchically repeating the process:



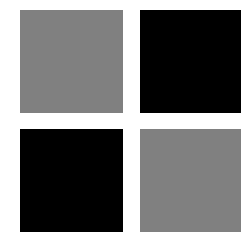
↓ Compress



↓ Compress



↓ Compress

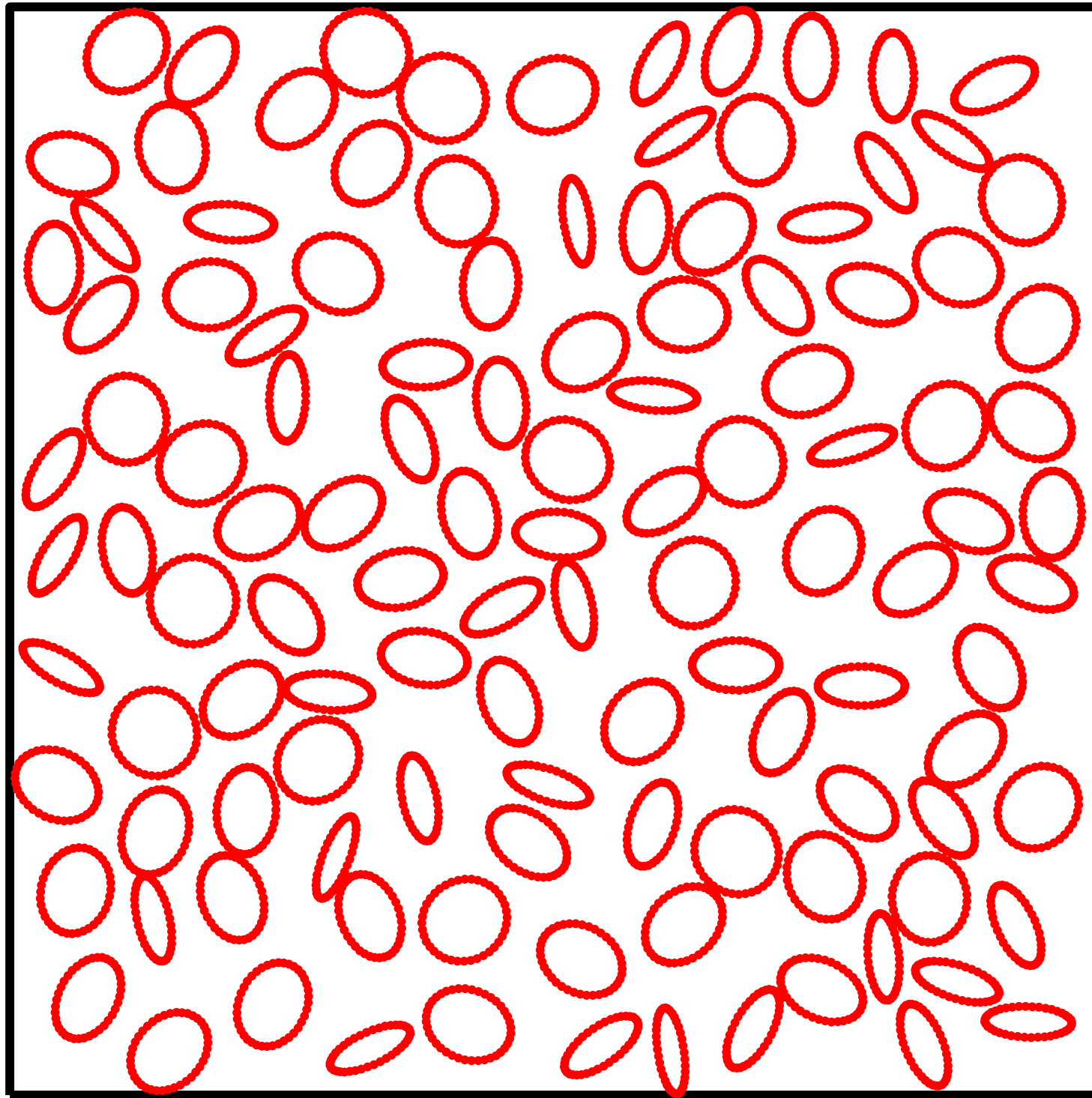


↗
Cluster

↗
Cluster

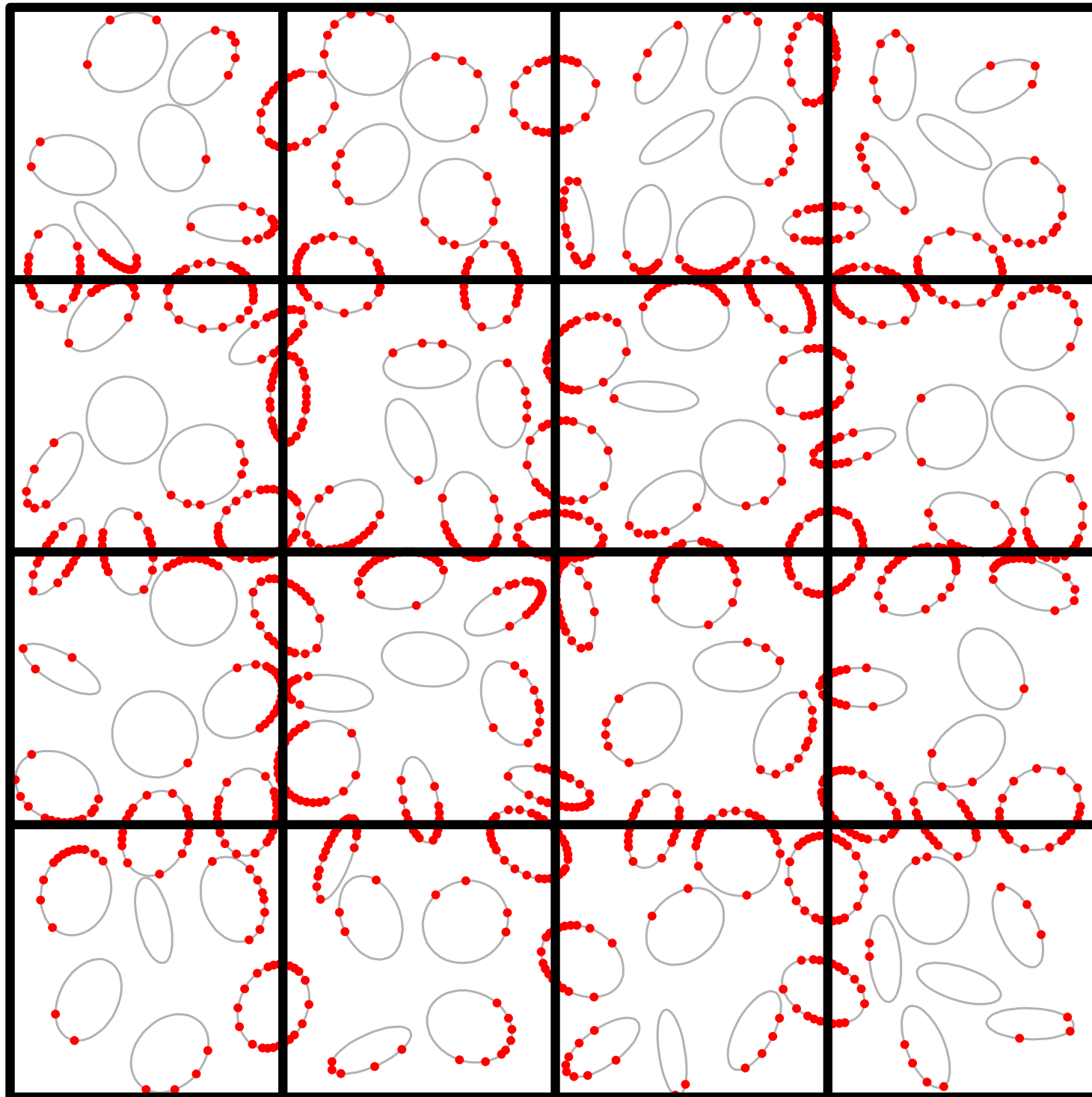
Example: Recursive skeletonization (a fast direct solver for integral equations)

Original set of points



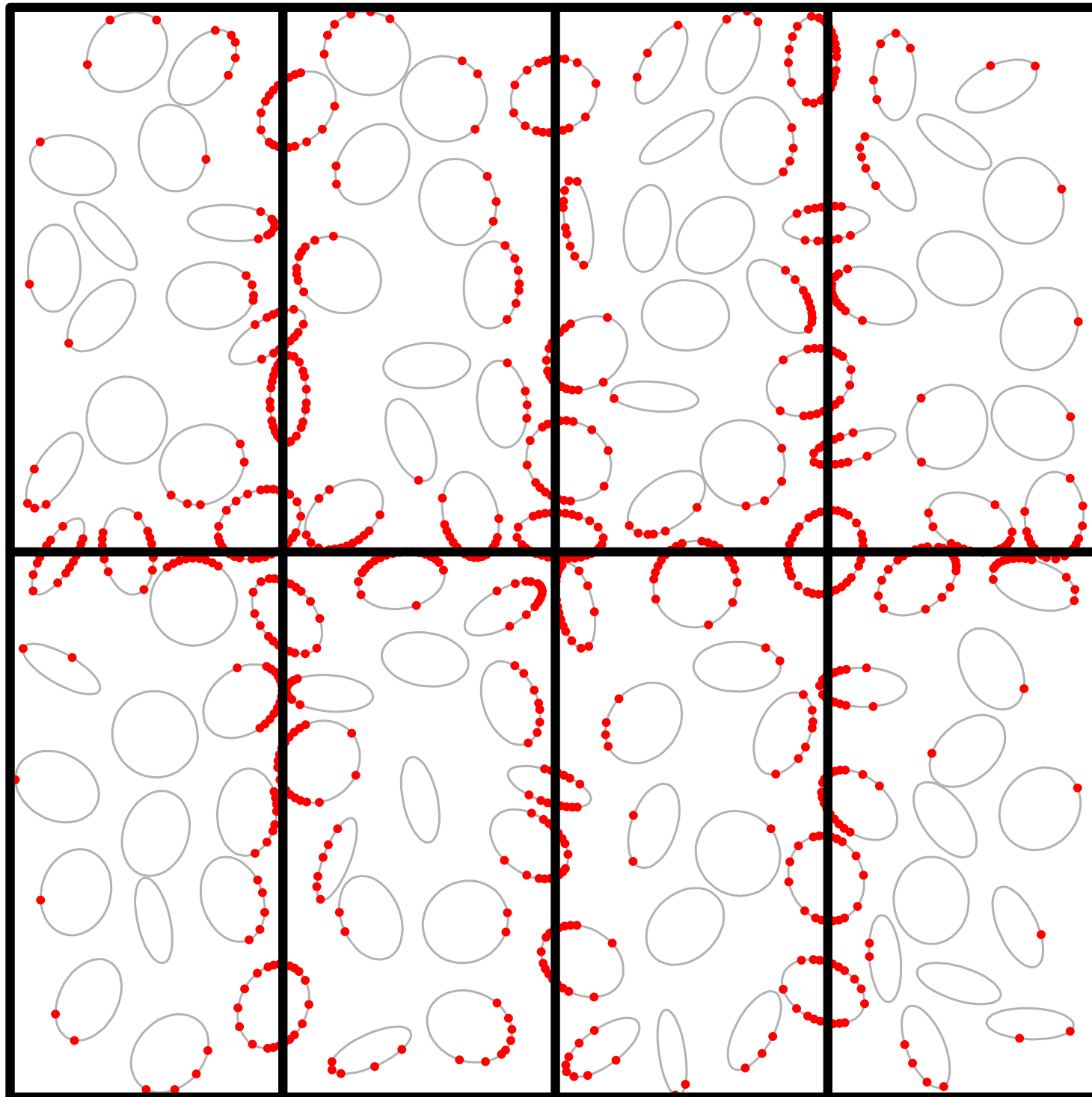
Example: Recursive skeletonization (a fast direct solver for integral equations)

Skeleton points on level 4, acc = 1.000e-09



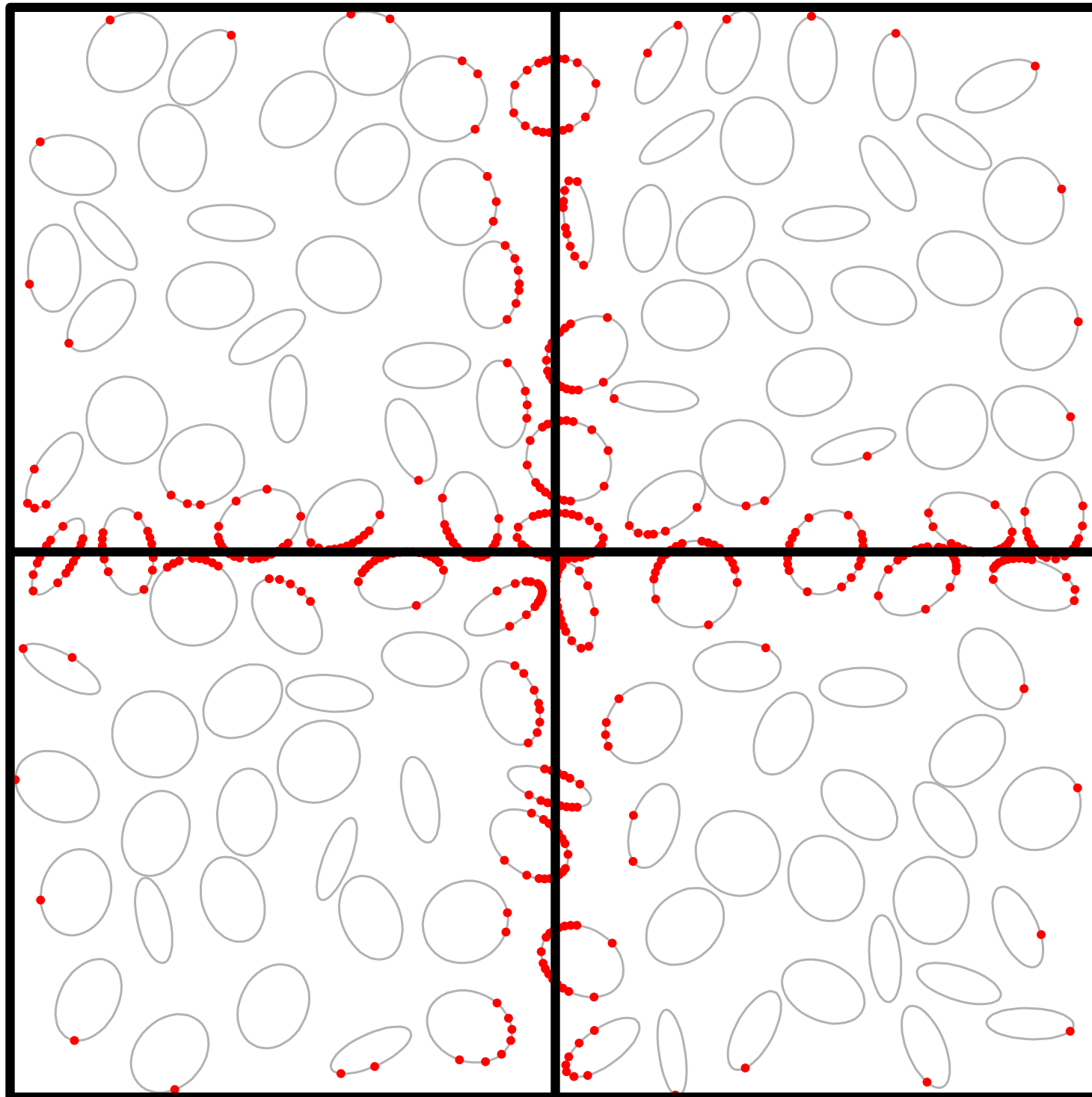
Example: Recursive skeletonization (a fast direct solver for integral equations)

Skeleton points on level 3, acc = 1.000e-09



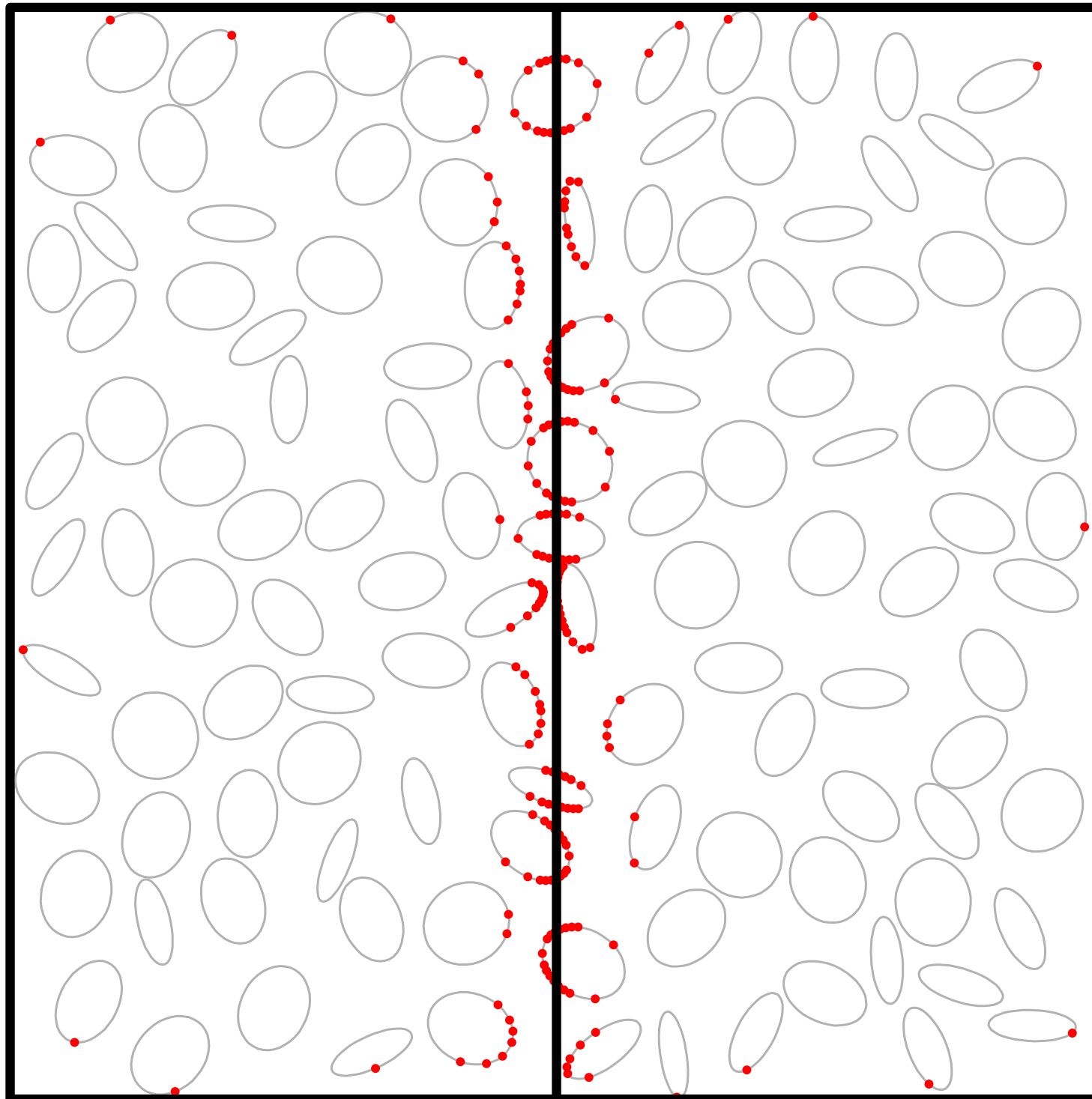
Example: Recursive skeletonization (a fast direct solver for integral equations)

Skeleton points on level 2, $\text{acc} = 1.000\text{e}-09$



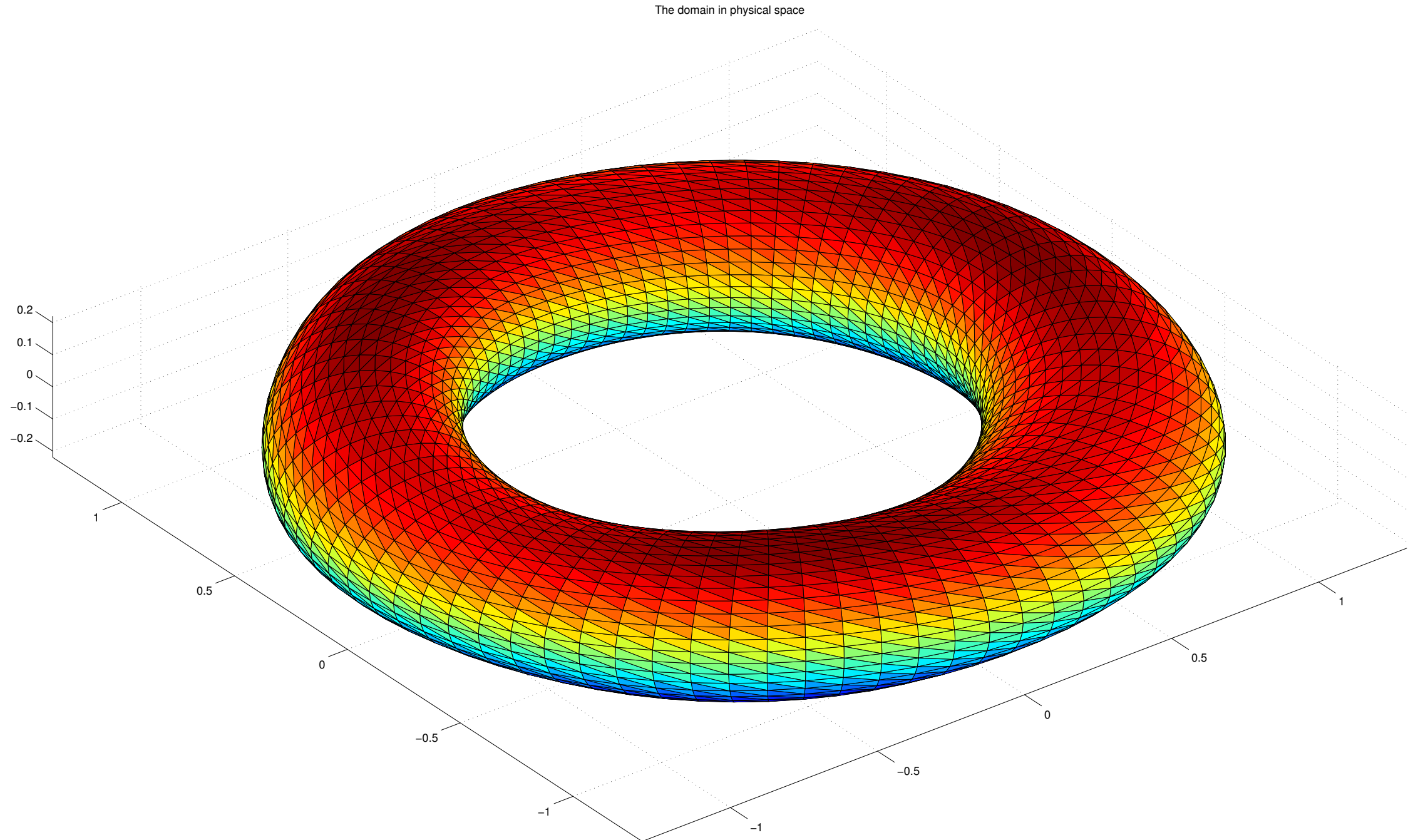
Example: Recursive skeletonization (a fast direct solver for integral equations)

Skeleton points on level 1, $\text{acc} = 1.000\text{e}-09$



Example: Recursive skeletonization (a fast direct solver for integral equations)

Now consider a surface in \mathbb{R}^3 :



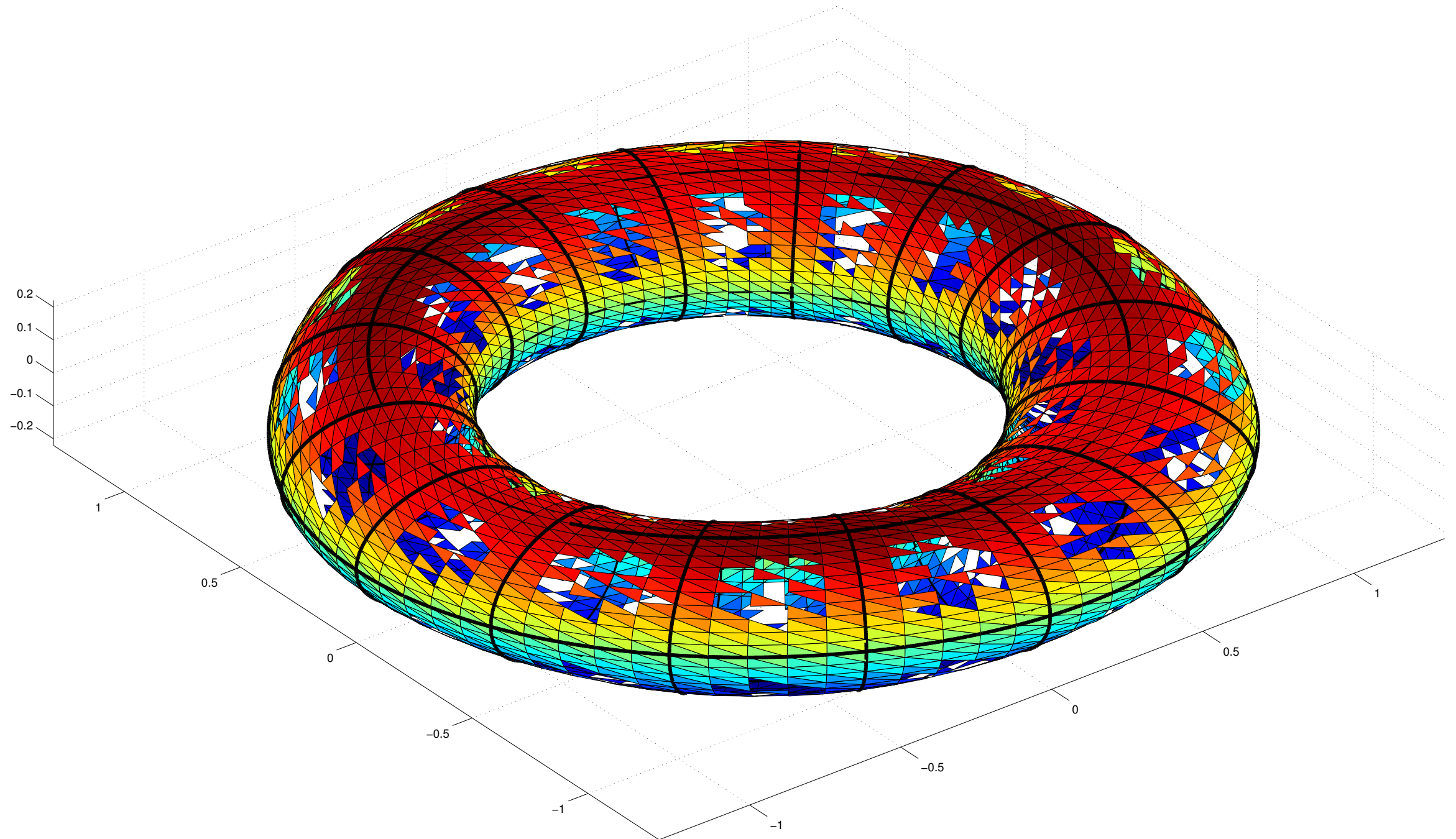
Let \mathbf{A} denote an $N \times N$ matrix arising upon discretizing a boundary integral operator

$$[Aq](\mathbf{x}) = q(\mathbf{x}) + \int_{\Gamma} \frac{1}{|\mathbf{x} - \mathbf{y}|} q(\mathbf{y}) dA(\mathbf{y}), \quad \mathbf{x} \in \Gamma,$$

where Γ is the “torus-like” domain shown (it is deformed to avoid rotational symmetry).

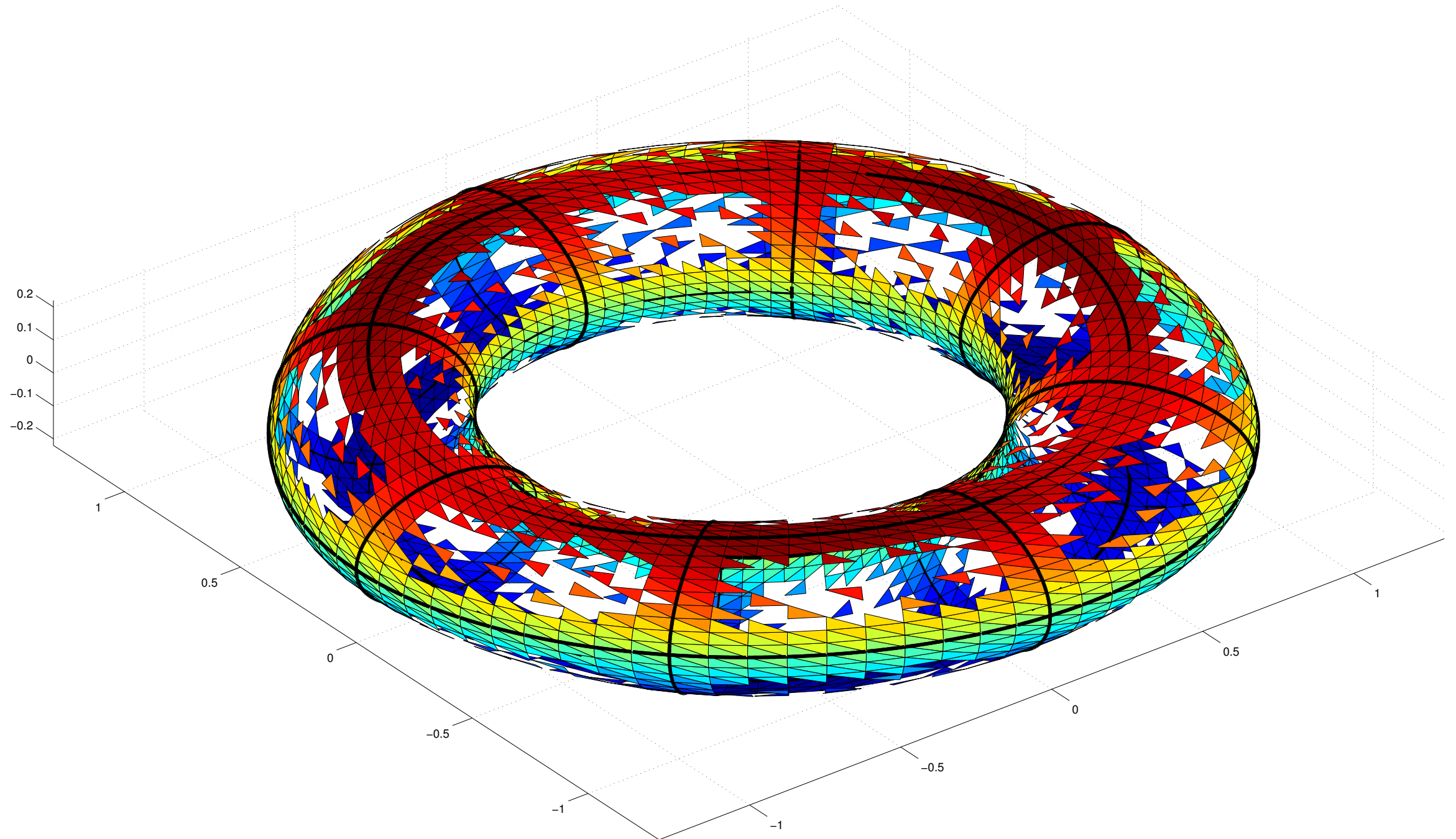
Example: Recursive skeletonization (a fast direct solver for integral equations)

The domain in physical space



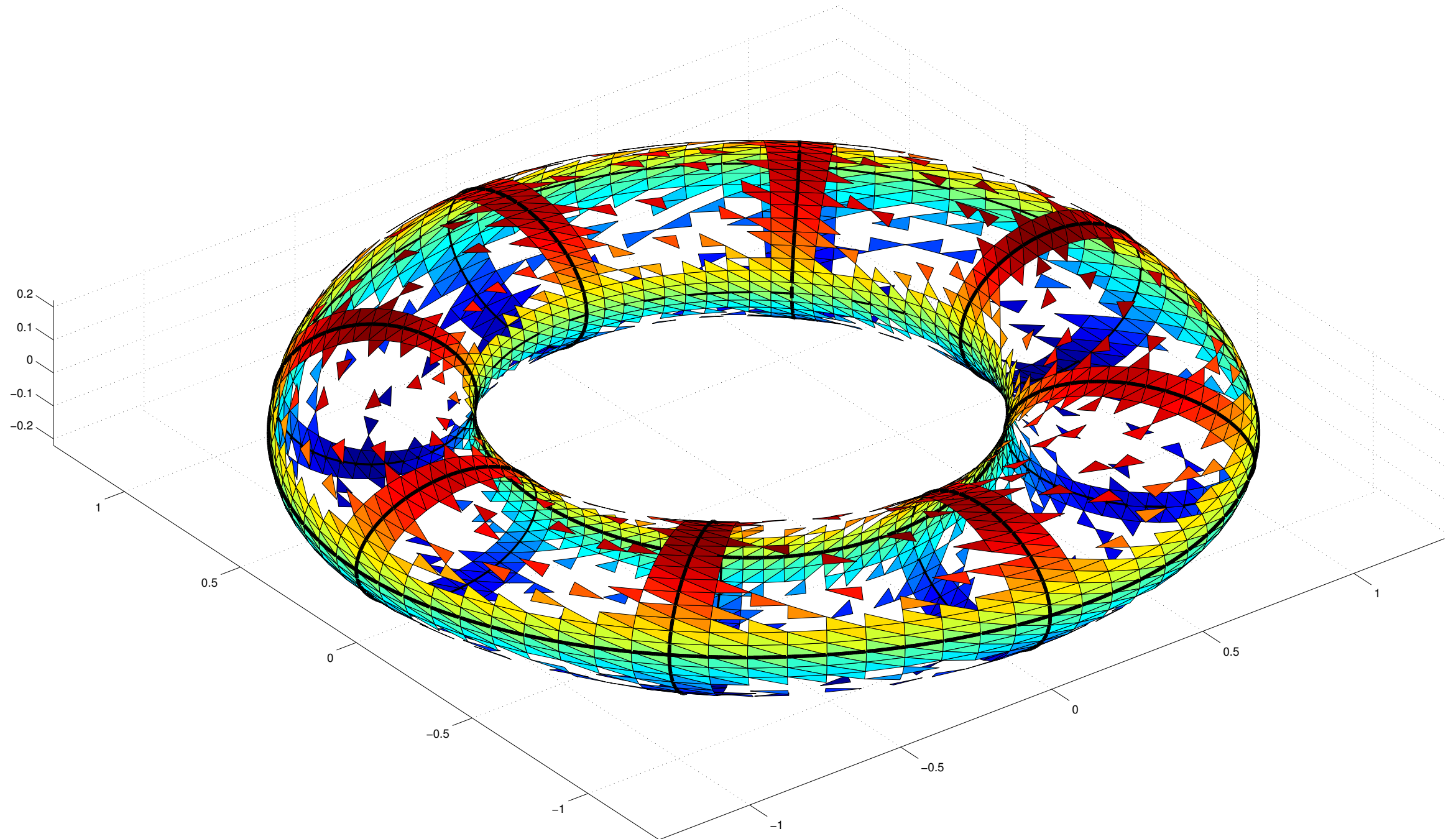
Example: Recursive skeletonization (a fast direct solver for integral equations)

The domain in physical space



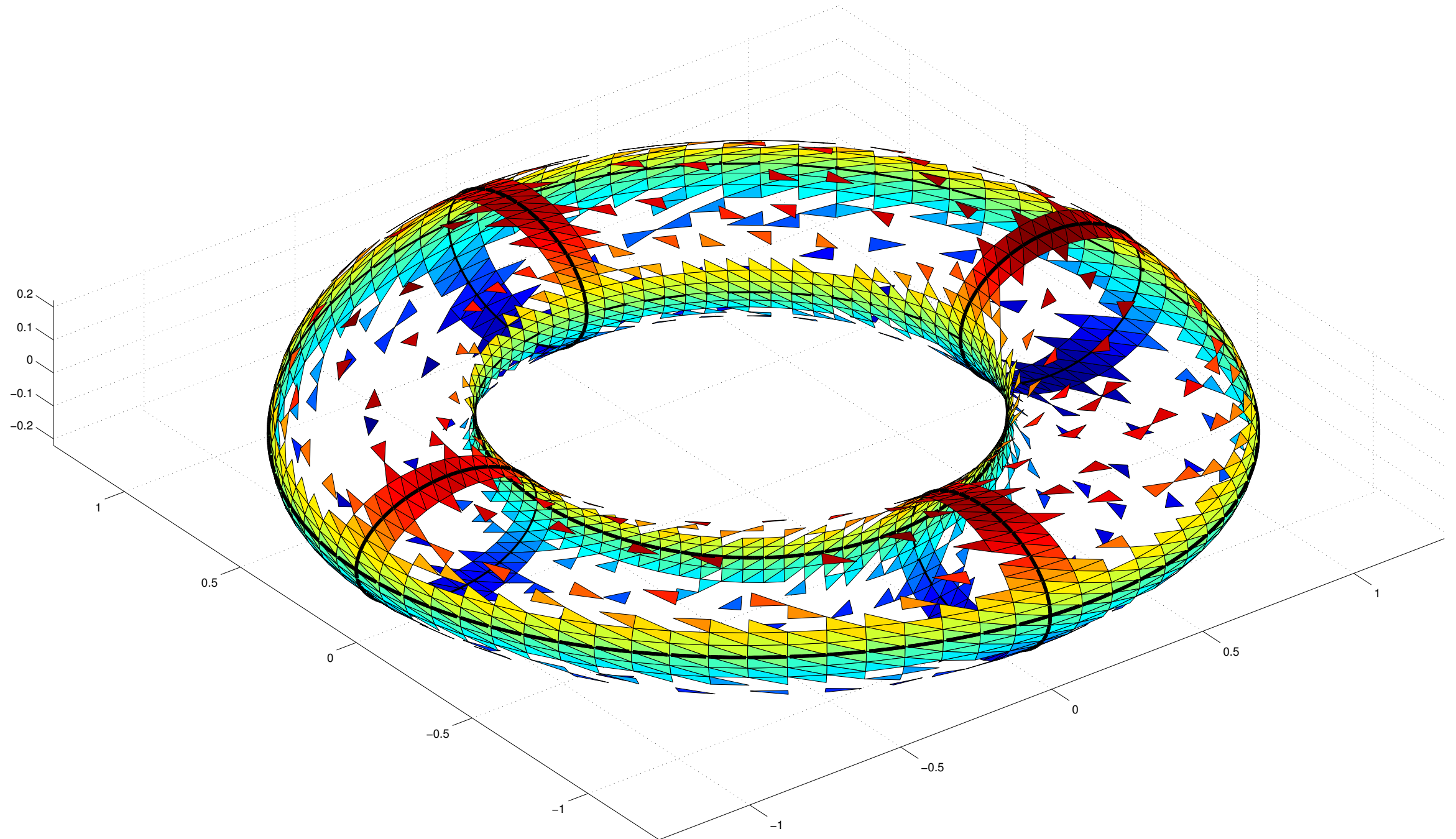
Example: Recursive skeletonization (a fast direct solver for integral equations)

The domain in physical space



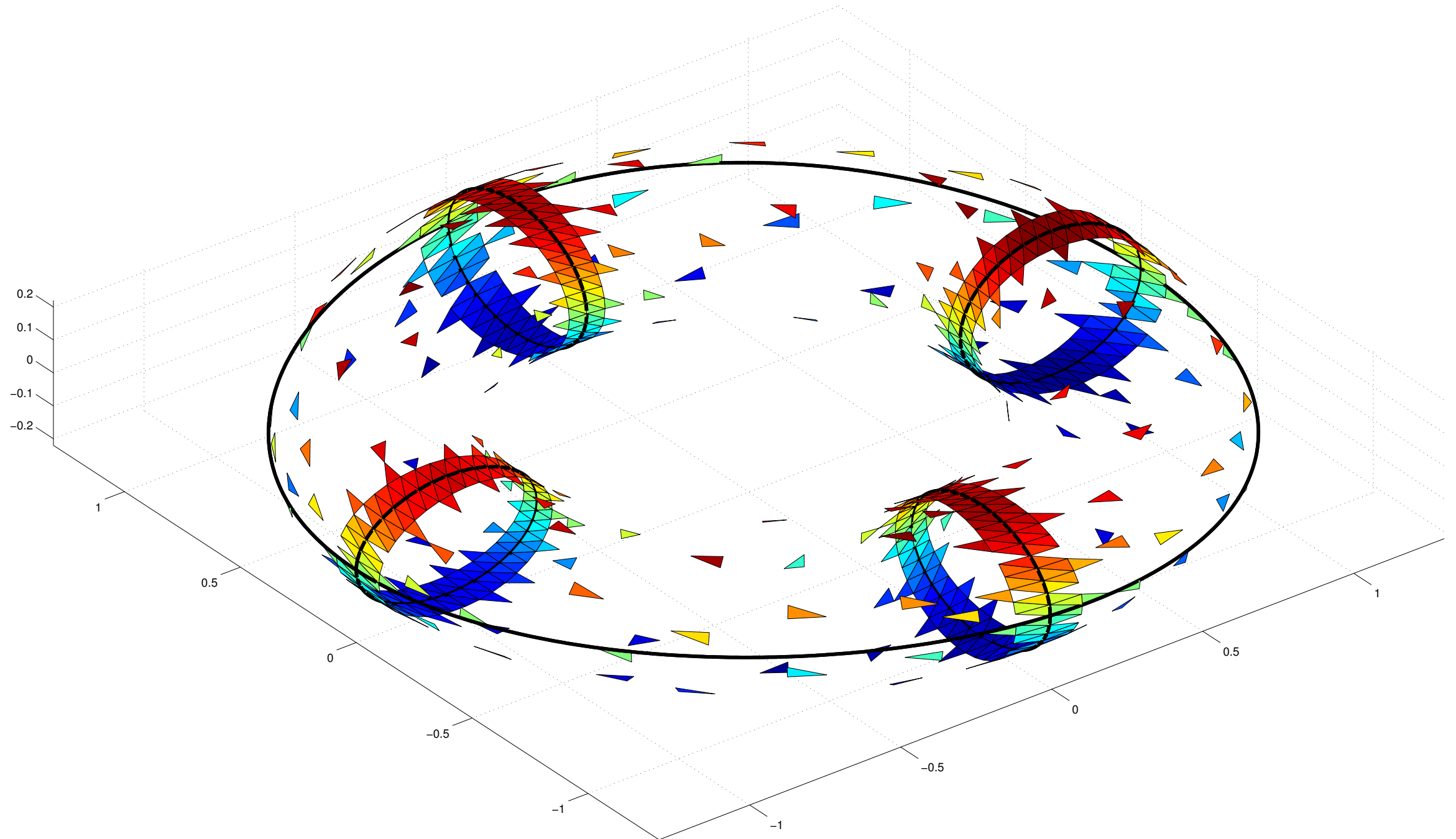
Example: Recursive skeletonization (a fast direct solver for integral equations)

The domain in physical space



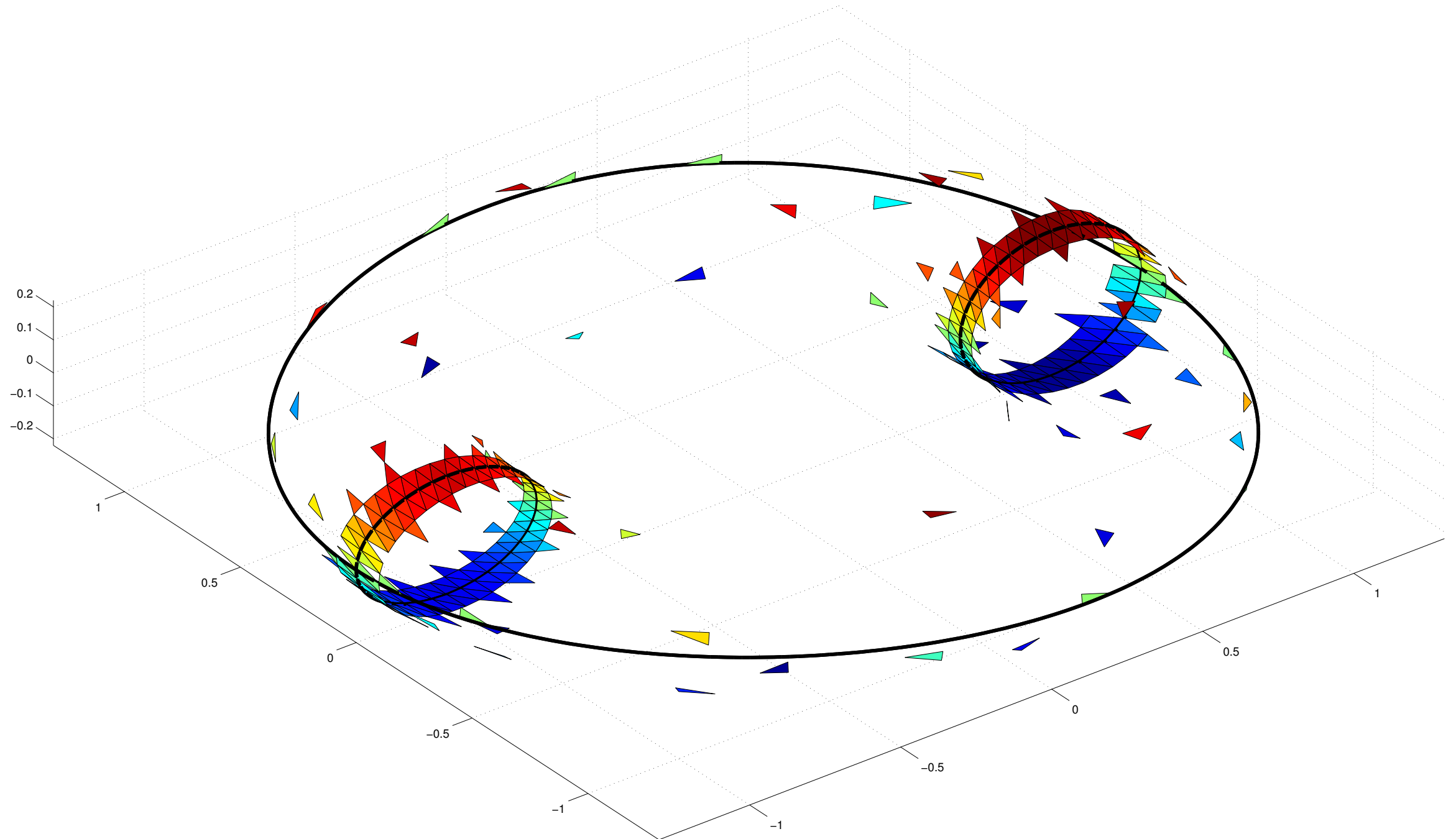
Example: Recursive skeletonization (a fast direct solver for integral equations)

The domain in physical space

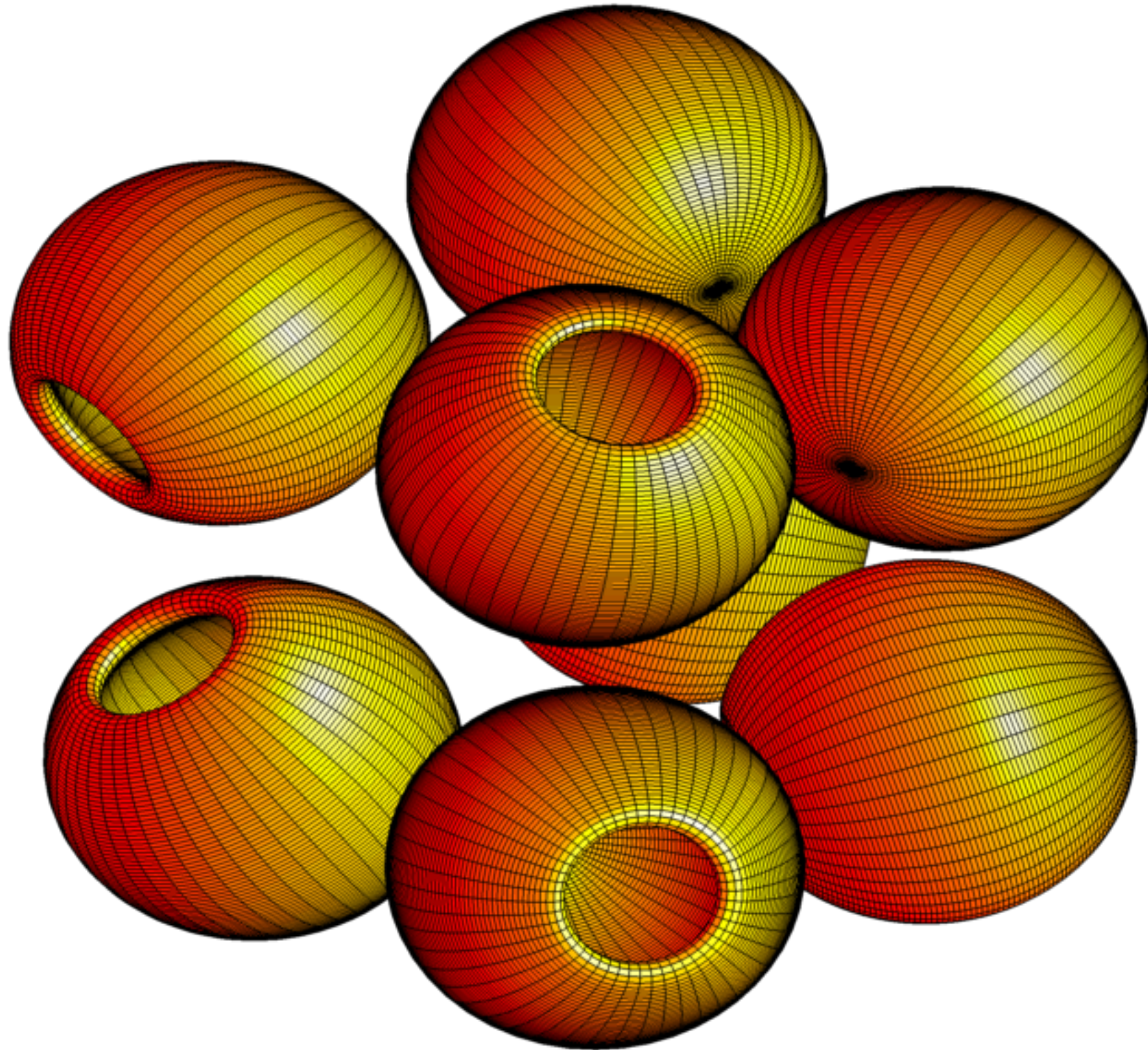


Example: Recursive skeletonization (a fast direct solver for integral equations)

The domain in physical space

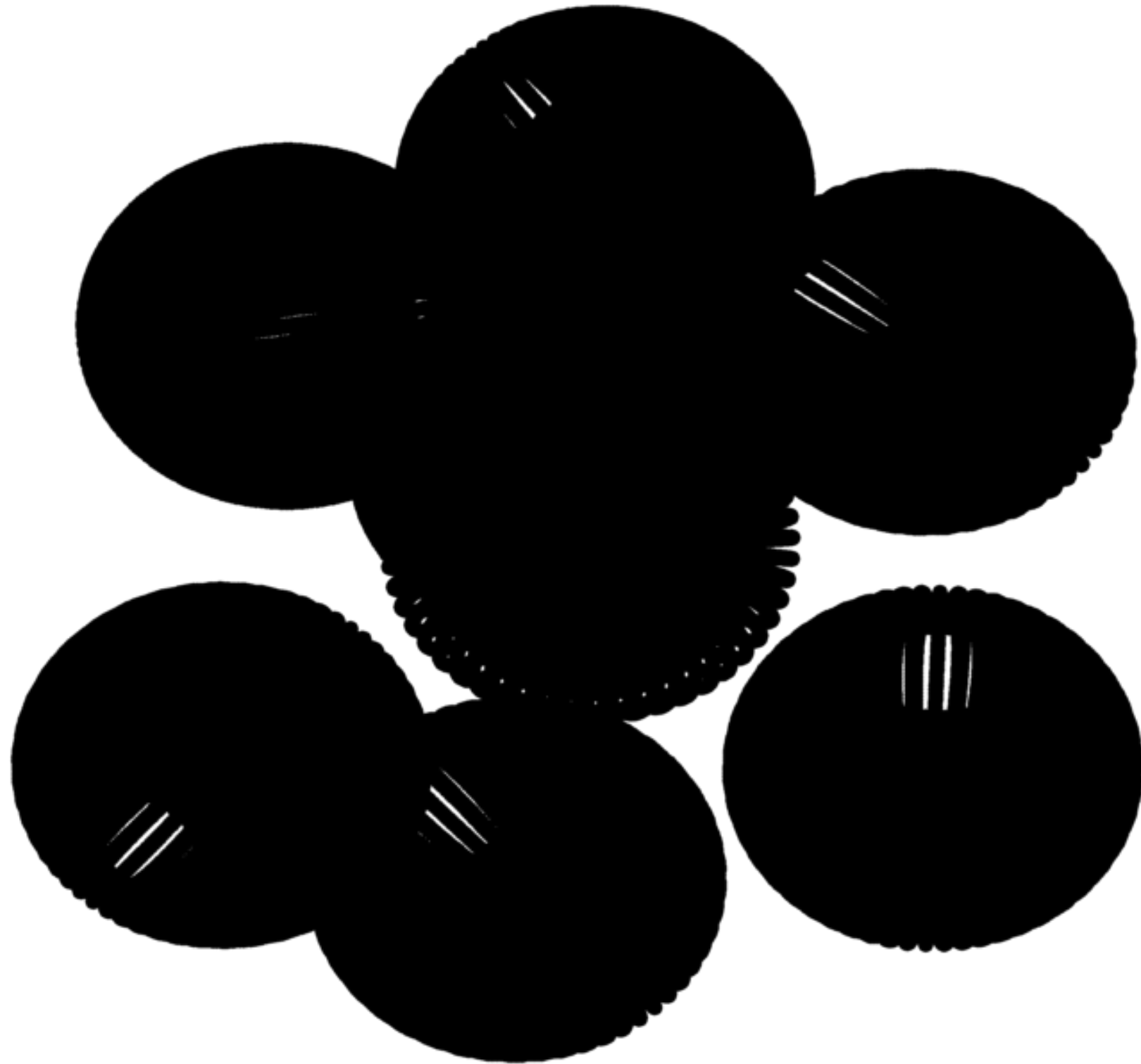


Example: Recursive skeletonization (a fast direct solver for integral equations)



Consider scattering from some multibody domain involving cavities.

Example: Recursive skeletonization (a fast direct solver for integral equations)



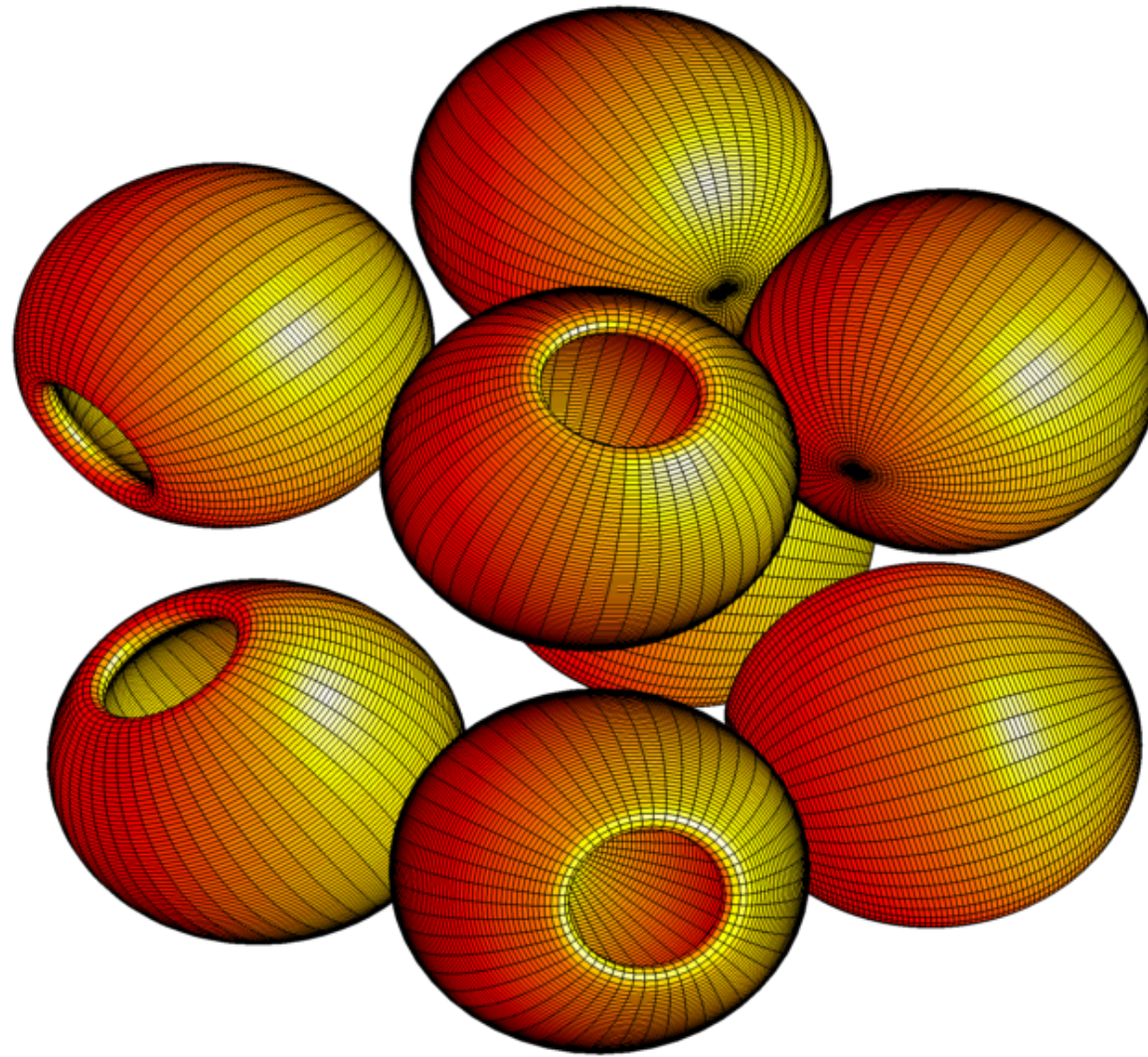
There are lots of discretization nodes involved. Very computationally intense!

Example: Recursive skeletonization (a fast direct solver for integral equations)



After local compression of each scatterer, the problem is much more tractable.

Example: Recursive skeletonization (a fast direct solver for integral equations)



Acoustic scattering on the exterior domain.

Each bowl is about 5λ .

A hybrid direct/iterative solver is used (a highly accurate scattering matrix is computed for each body).

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

Accuracy 10^{-7} took 27h.

Example: Recursive skeletonization (a fast direct solver for integral equations)

BIEs on rotationally symmetric bodies (2014, with S. Hao and P. Young)

N	N_{body}	T_{fmm}	I_{GMRES} (precond /no precondition)	T_{total} (precond /no precondition)	E_{∞}^{rel}
10000	50 × 25	1.23e+00	21 /358	2.70e+01 /4.49e+02	4.414e-04
20000	100 × 25	3.90e+00	21 /331	8.57e+01 /1.25e+03	4.917e-04
40000	200 × 25	6.81e+00	21 /197	1.62e+02 /1.18e+03	4.885e-04
80000	400 × 25	1.36e+01	21 / 78	3.51e+02 /1.06e+03	4.943e-04
20400	50 × 51	4.08e+00	21 /473	8.67e+01 /1.99e+03	1.033e-04
40800	100 × 51	7.20e+00	21 /442	1.56e+02 /3.17e+03	3.212e-05
81600	200 × 51	1.35e+01	21 /198	2.99e+02 /2.59e+03	9.460e-06
163200	400 × 51	2.50e+01	21 /102	5.85e+02 /2.62e+03	1.011e-05
40400	50 × 101	7.21e+00	21 /483	1.53e+02 /3.52e+03	1.100e-04
80800	100 × 101	1.34e+01	22 /452	2.99e+02 /6.31e+03	3.972e-05
161600	200 × 101	2.55e+01	22 /199	5.80e+02 /5.12e+03	2.330e-06
323200	400 × 101	5.36e+01	22 /112	1.25e+03 /5.84e+03	3.035e-06

Exterior Laplace problem solved on the multibody bowl domain with and without preconditioner.

Example: Recursive skeletonization (a fast direct solver for integral equations)

N	N_{body}	$T_{\text{precompute}}$	l_{GMRES}	T_{solve}	E_{∞}^{rel}
80800	100×101	6.54e-01	62	5.17e+03	1.555e-03
161600	200×101	1.82e+00	63	9.88e+03	1.518e-04
323200	400×101	6.46e+00	64	2.19e+04	3.813e-04
160800	100×201	1.09e+00	63	9.95e+03	1.861e-03
321600	200×201	3.00e+00	64	2.19e+04	2.235e-05
643200	400×201	1.09e+01	64	4.11e+04	8.145e-06
641600	200×401	5.02e+00	64	4.07e+04	2.485e-05
1283200	400×401	1.98e+01	65	9.75e+04	6.884e-07

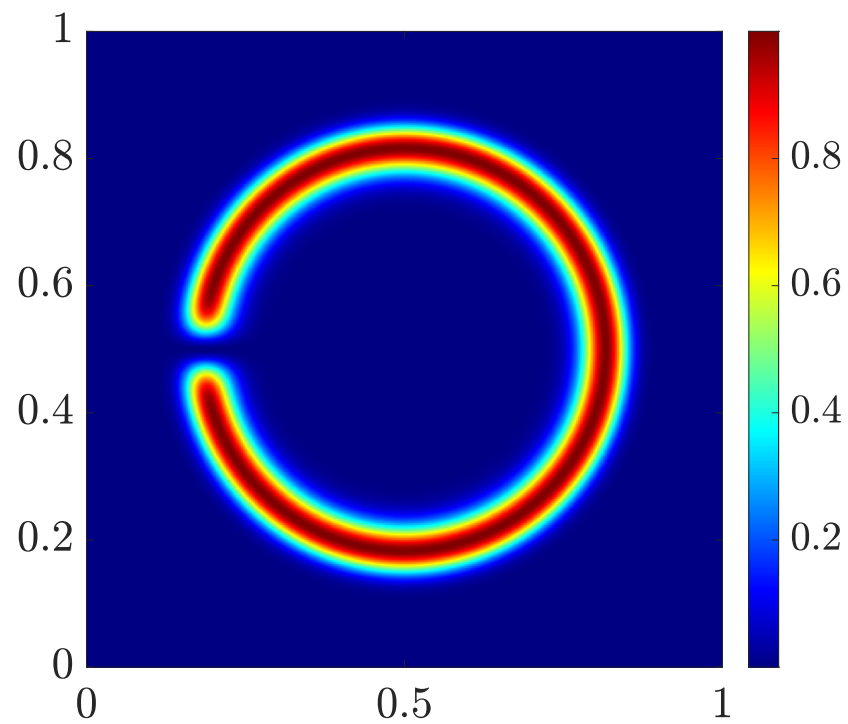
*Exterior **Helmholtz** problem solved on multibody bowl domain.*

Each bowl is 5 wavelength in diameter.

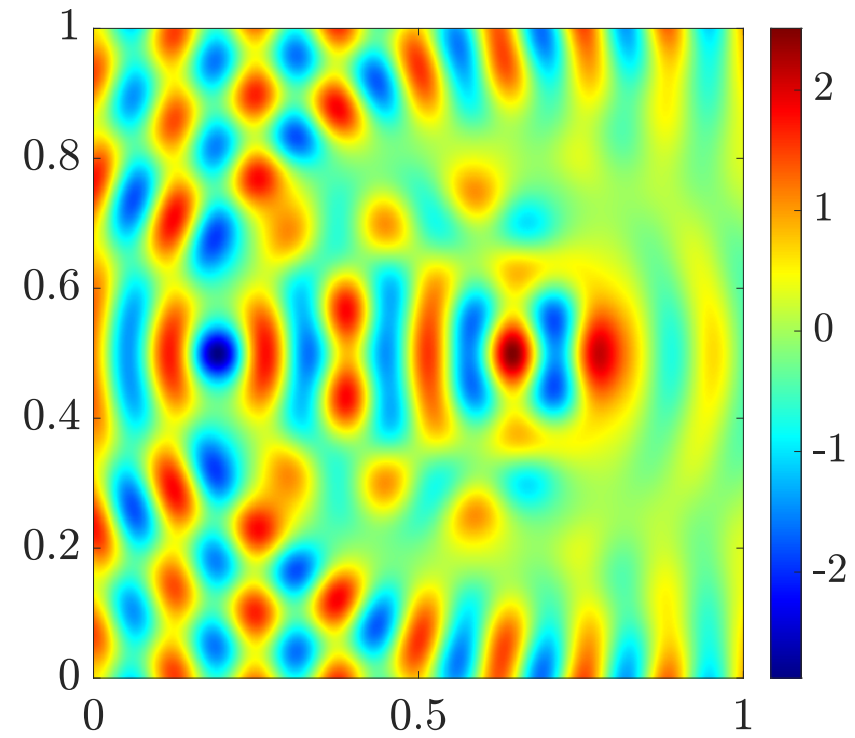
We do not give timings for standard iterative methods since in this example, they typically did not converge at all (even though the BIE is a 2nd kind Fredholm equation).

Example: Recursive skeletonization (a fast direct solver for integral equations)

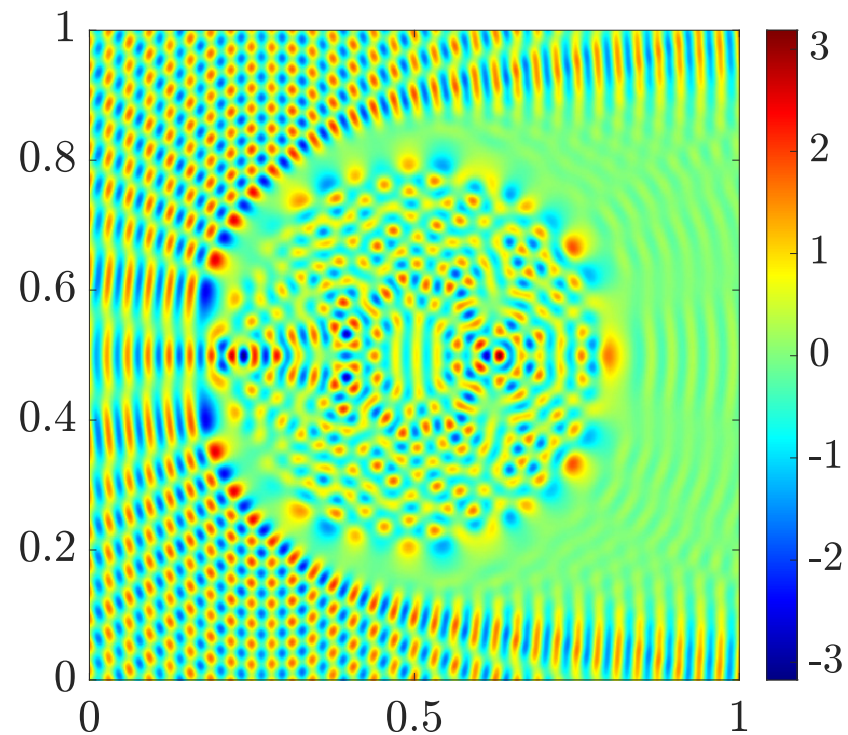
Lippmann-Schwinger in the plane for variable coefficient problem. High order quadrature rule.



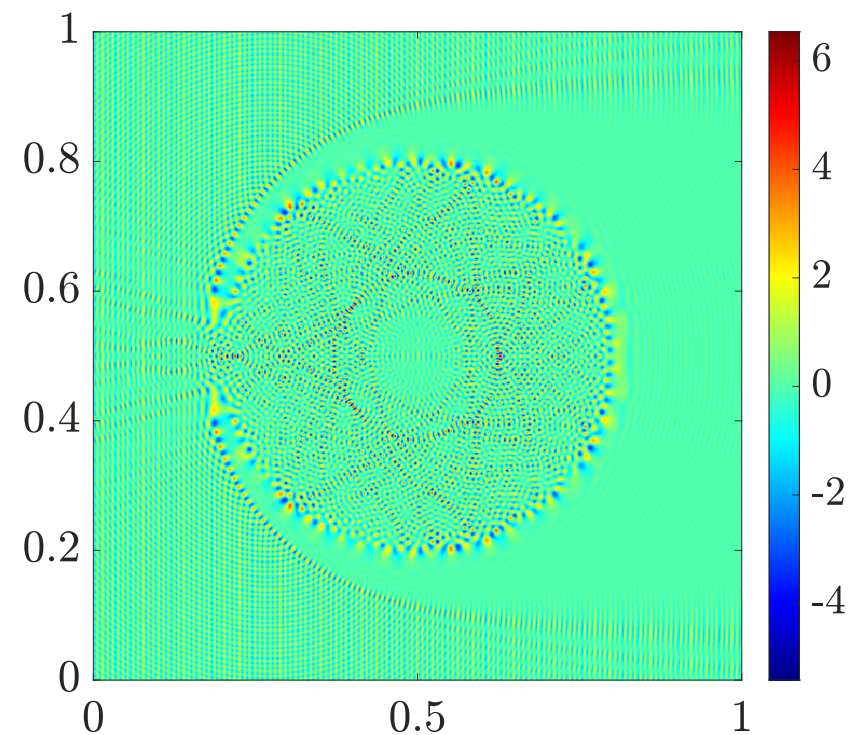
The scattering potential



$\kappa = 50$



$\kappa = 201$



$\kappa = 804$

Example: Recursive skeletonization (a fast direct solver for integral equations)

Fixed 10 points per wavelength.

Direct solver is run at accuracy 10^{-3} and used as a preconditioner.

Weak admissibility is used.

N	κ	T_{build}	T_{inv}	T_{gmres}	mem	iter	res
6400	50.27	0.23	0.24	0.20	0.04	4	6.97e-11
25600	100.53	0.65	0.99	0.62	0.21	5	6.16e-12
102400	201.06	2.26	4.36	2.49	1.01	6	1.04e-12
409600	402.12	14.91	20.06	9.78	4.67	6	3.23e-11
1638400	804.25	99.01	91.37	56.13	21.16	9	8.12e-12
6553600	1608.50	430.60	398.88	330.91	94.63	13	3.93e-11
26214400	3216.99	3102.09	2024.16	2698.53	418.37	22	3.30e-11

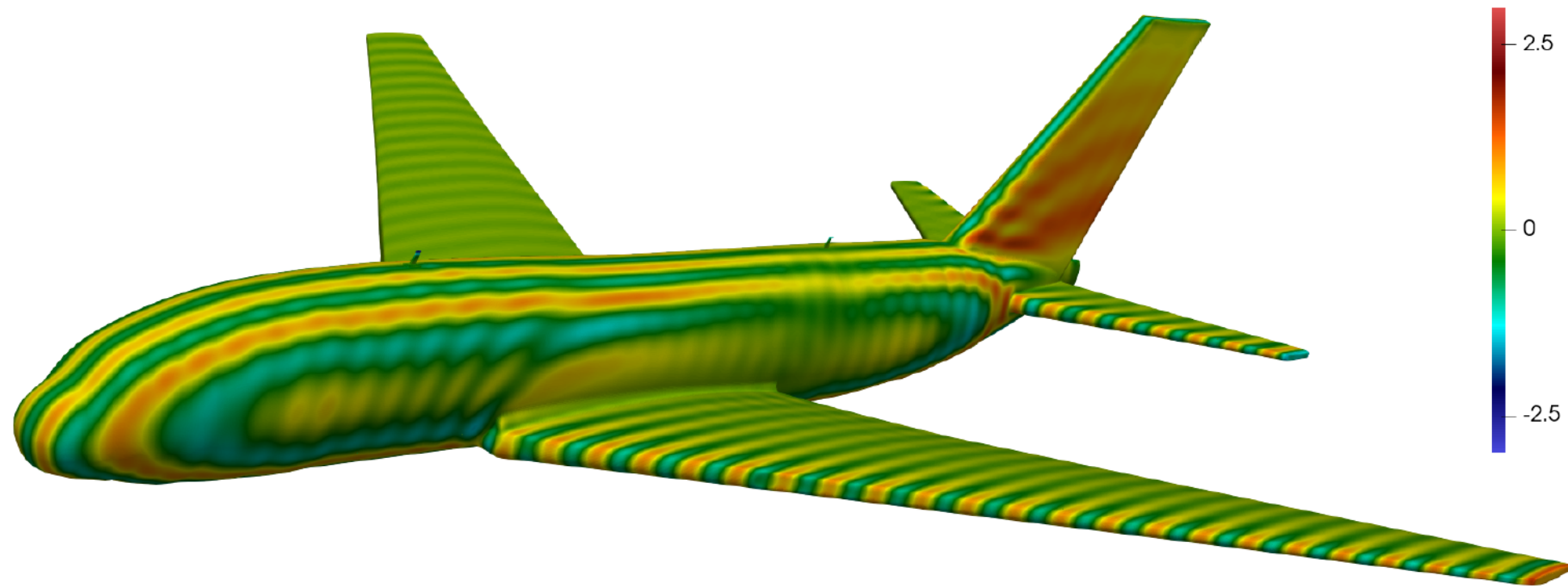
The largest experiment is over 500λ in diameter: Less than 3h total run time.

Hardware: Workstation with dual Intel Xeon Gold 6254 (18 cores at 3.1GHz base frequency).

Joint with Abi Gopal, Advances in Computational Mathematics, 48(4), pp. 1 - 31, 2022.

Example: Recursive skeletonization (a fast direct solver for integral equations)

Acoustic scattering (for now, objective is electro-magnetics, of course):



$$50\lambda \times 50\lambda \times 14\lambda$$

Results from sequential (except for dense linear algebra) Matlab code:

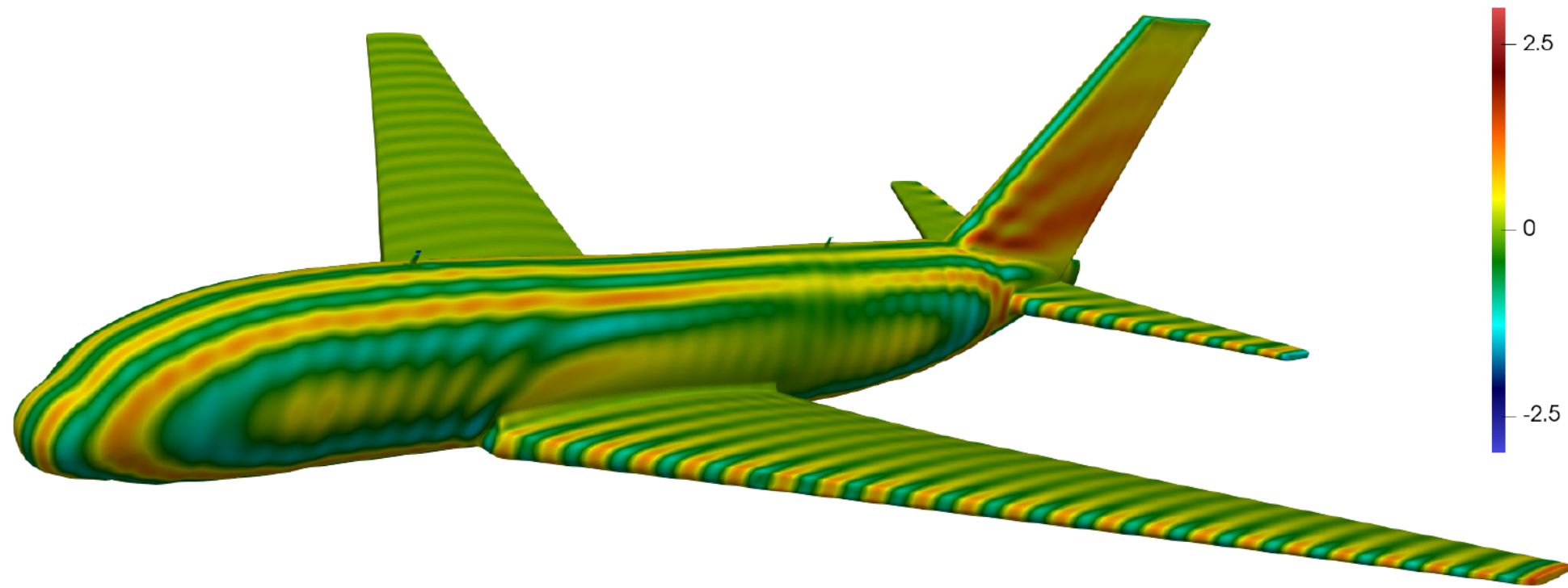
$N = 1.2\text{M}$. Factorization time = 4h. Solve time = 30s. Memory req = 460GB. Precision = 10^{-3} .

D. Sushnikova, L. Greengard, M. O'Neil, M. Rachh; arXiv:2201.07325.

Current work: Parallelize, and develop HPC implementation. *With Chao Chen.*

Example: Recursive skeletonization (a fast direct solver for integral equations)

Acoustic scattering (for now, objective is electro-magnetics, of course):



$$50\lambda \times 50\lambda \times 14\lambda$$

Results from sequential (except for dense linear algebra) Matlab code:

$N = 1.2\text{M}$. Factorization time = 4h. Solve time = 30s. Memory req = 460GB. Precision = 10^{-3} .

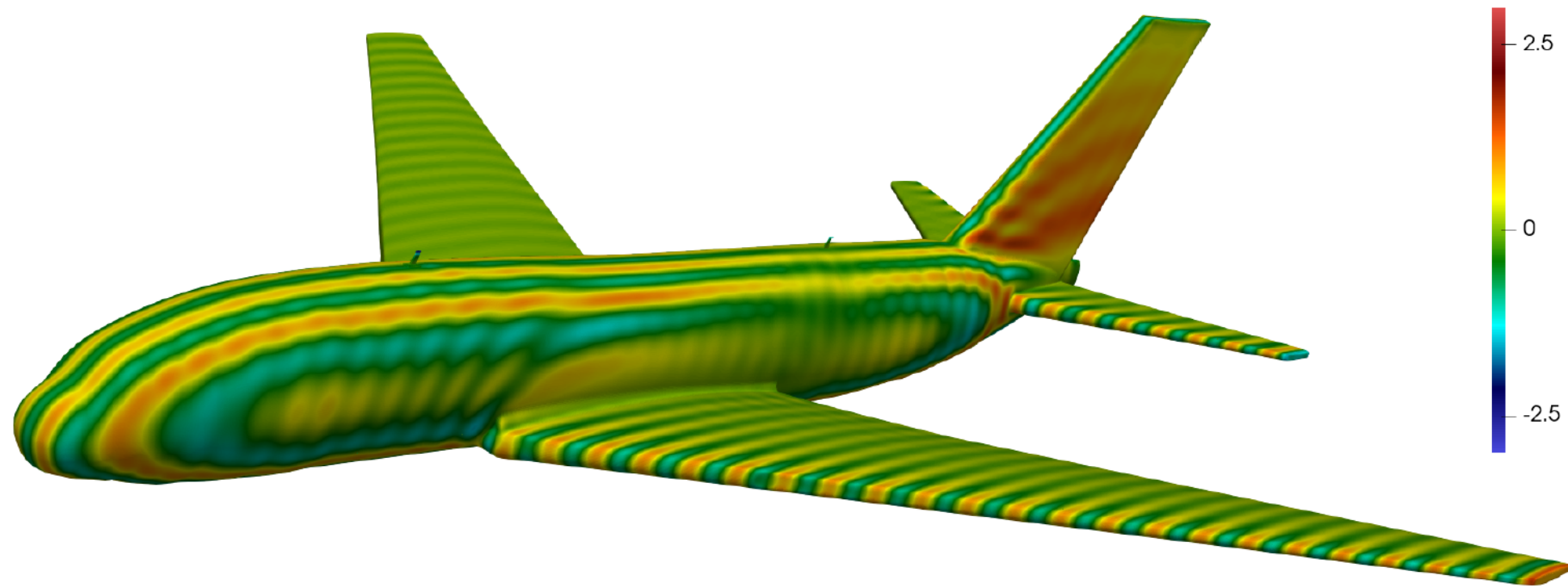
D. Sushnikova, L. Greengard, M. O'Neil, M. Rachh; arXiv:2201.07325.

Current work: Parallelize, and develop HPC implementation. *With Chao Chen.*

Question: Can you combine rank-structure & least squares problems?

Example: Recursive skeletonization (a fast direct solver for integral equations)

Acoustic scattering (for now, objective is electro-magnetics, of course):



$$50\lambda \times 50\lambda \times 14\lambda$$

Results from sequential (except for dense linear algebra) Matlab code:

$N = 1.2\text{M}$. Factorization time = 4h. Solve time = 30s. Memory req = 460GB. Precision = 10^{-3} .

D. Sushnikova, L. Greengard, M. O'Neil, M. Rachh; arXiv:2201.07325.

Current work: Parallelize, and develop HPC implementation. *With Chao Chen.*

Question: Can you combine rank-structure & least squares problems?

Answer appears to be yes! Current work by Heather Wilber and Ethan Epperly.

Alternatively: Build numerical “scattering matrices” locally using least squares.

Outline of talk

- The interpolatory and CUR decompositions — what are they?
- Applications of IDs in solving PDEs and integral equations.
- **Efficient algorithms for computing an interpolatory decomposition.**
- PDE applications revisited: How to compress a global operator. *[Time permitting]*

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix \mathbf{A} , and a target rank k , find an index vector J_S of length k , and a matrix \mathbf{Z} of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_S)\mathbf{Z}.$$

We additionally require \mathbf{Z} to contain the $k \times k$ identity matrix as a submatrix.

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix \mathbf{A} , and a target rank k , find an index vector J_s of length k , and a matrix \mathbf{Z} of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_s) \mathbf{Z}.$$

We additionally require \mathbf{Z} to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of \mathbf{A} . This results in a factorization

$$\begin{array}{ccc} \mathbf{A}(:, J) \approx & \mathbf{Q} & \left[\begin{array}{cc} \mathbf{R}_{ss} & \mathbf{R}_{sr} \end{array} \right] \\ m \times n & m \times k & \begin{array}{cc} k \times k & k \times (n - k) \end{array} \end{array}$$

Set $J_s = J(1 : k)$, and pull out the factor \mathbf{R}_{ss} to form the column-ID:

$$\mathbf{A}(:, J) \approx \mathbf{Q} \mathbf{R}_{ss} \left[\mathbf{I} \quad \mathbf{R}_{ss}^{-1} \mathbf{R}_{sr} \right] = \mathbf{A}(:, J_s) \mathbf{Z}(:, J).$$

As previously seen in Anil Damle's talk on Tuesday!

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix \mathbf{A} , and a target rank k , find an index vector J_s of length k , and a matrix \mathbf{Z} of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_s) \mathbf{Z}.$$

We additionally require \mathbf{Z} to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of \mathbf{A} . This results in a factorization

$$\begin{array}{ccc} \mathbf{A}(:, J) \approx & \mathbf{Q} & \left[\begin{array}{cc} \mathbf{R}_{ss} & \mathbf{R}_{sr} \end{array} \right] \\ m \times n & m \times k & \begin{array}{cc} k \times k & k \times (n - k) \end{array} \end{array}$$

Set $J_s = J(1 : k)$, and pull out the factor \mathbf{R}_{ss} to form the column-ID:

$$\mathbf{A}(:, J) \approx \mathbf{Q} \mathbf{R}_{ss} \left[\mathbf{I} \quad \mathbf{R}_{ss}^{-1} \mathbf{R}_{sr} \right] = \mathbf{A}(:, J_s) \mathbf{Z}(:, J).$$

Notes:

- Orthonormality must be maintained *scrupulously*. Use Householder or “double” Gram-Schmidt.
- CPQR can in principle fail (e.g. the “Kahan counter example”), but in practice it works very well.
- Reasonably computationally efficient for matrices that fit in RAM.
- Sophisticated versions of CPQR have been developed that *guarantee* close to optimal column selection, as well as bounding all elements of $\mathbf{R}_{ss}^{-1} \mathbf{R}_{sr}$. (*Gu & Eisenstat SISC 1996*)

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix \mathbf{A} , and a target rank k , find an index vector J_s of length k , and a matrix \mathbf{Z} of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}.$$

We additionally require \mathbf{Z} to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of \mathbf{A} . This results in a factorization

$$\begin{array}{ccc} \mathbf{A}(:, J) \approx & \mathbf{Q} & \left[\begin{array}{cc} \mathbf{R}_{ss} & \mathbf{R}_{sr} \end{array} \right] \\ m \times n & m \times k & \begin{array}{cc} k \times k & k \times (n - k) \end{array} \end{array}$$

Set $J_s = J(1 : k)$, and pull out the factor \mathbf{R}_{ss} to form the column-ID:

$$\mathbf{A}(:, J) \approx \mathbf{Q}\mathbf{R}_{ss} \left[\mathbf{I} \quad \mathbf{R}_{ss}^{-1}\mathbf{R}_{sr} \right] = \mathbf{A}(:, J_s)\mathbf{Z}(:, J).$$

Questions:

- Can you efficiently process large matrices that do not fit in fast memory?
- Can you efficiently process huge *sparse* matrices?
- Can you improve on the practical speed of CPQR? Even on the $O(mnk)$ complexity?

The column selection problem — through a *sketch*

Simple fact: Let \mathbf{A} be an $m \times n$ matrix of **exact** rank k . Suppose:

(1) We have by some means computed a factorization

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{E} & \mathbf{F}. \\ m \times n & & m \times k & k \times n \end{array}$$

(2) We have solved the column selection problem for \mathbf{F} , so that

$$\begin{array}{ccccc} \mathbf{F} & = & \mathbf{F}(:, J_S) & \mathbf{Z}. \\ k \times n & & k \times k & k \times n \end{array}$$

Then, *automatically*, we have also solved the column selection problem for \mathbf{A} :

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{A}(:, J_S) & \mathbf{Z}. \\ m \times n & & m \times k & k \times n \end{array}$$

The column selection problem — through a *sketch*

Simple fact: Let \mathbf{A} be an $m \times n$ matrix of **exact** rank k . Suppose:

(1) We have by some means computed a factorization

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{E} & \mathbf{F}. \\ m \times n & & m \times k & k \times n \end{array}$$

(2) We have solved the column selection problem for \mathbf{F} , so that

$$\begin{array}{ccccc} \mathbf{F} & = & \mathbf{F}(:, J_S) & \mathbf{Z}. \\ k \times n & & k \times k & k \times n \end{array}$$

Then, *automatically*, we have also solved the column selection problem for \mathbf{A} :

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{A}(:, J_S) & \mathbf{Z}. \\ m \times n & & m \times k & k \times n \end{array}$$

Proof: Assume that

$$(6) \quad \mathbf{A} = \mathbf{E}\mathbf{F}$$

and that

$$(7) \quad \mathbf{F} = \mathbf{F}(:, J_S)\mathbf{Z}.$$

Then

$$\mathbf{A}(:, J_S)\mathbf{Z} \stackrel{(2)}{=} \mathbf{E}\mathbf{F}(:, J_S)\mathbf{Z} \stackrel{(3)}{=} \mathbf{E}\mathbf{F} = \mathbf{A}.$$

The column selection problem — through a *sketch*

Simple fact: Let \mathbf{A} be an $m \times n$ matrix of **exact** rank k . Suppose:

(1) We have by some means computed a factorization

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{E} & \mathbf{F}. \\ m \times n & & m \times k & k \times n \end{array}$$

(2) We have solved the column selection problem for \mathbf{F} , so that

$$\begin{array}{ccccc} \mathbf{F} & = & \mathbf{F}(:, J_S) & \mathbf{Z}. \\ k \times n & & k \times k & k \times n \end{array}$$

Then, *automatically*, we have also solved the column selection problem for \mathbf{A} :

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{A}(:, J_S) & \mathbf{Z}. \\ m \times n & & m \times k & k \times n \end{array}$$

Question: How do you find a $k \times n$ matrix \mathbf{F} such that $\mathbf{A} = \mathbf{E}\mathbf{F}$ for some \mathbf{E} ?

Hint: Cora Cartis talk on Monday ...

The column selection problem — through a *sketch*

Simple fact: Let \mathbf{A} be an $m \times n$ matrix of **exact** rank k . Suppose:

(1) We have by some means computed a factorization

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{E} & \mathbf{F}. \\ m \times n & & m \times k & k \times n \end{array}$$

(2) We have solved the column selection problem for \mathbf{F} , so that

$$\begin{array}{ccccc} \mathbf{F} & = & \mathbf{F}(:, J_S) & \mathbf{Z}. \\ k \times n & & k \times k & k \times n \end{array}$$

Then, *automatically*, we have also solved the column selection problem for \mathbf{A} :

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{A}(:, J_S) & \mathbf{Z}. \\ m \times n & & m \times k & k \times n \end{array}$$

Question: How do you find a $k \times n$ matrix \mathbf{F} such that $\mathbf{A} = \mathbf{E}\mathbf{F}$ for some \mathbf{E} ?

Randomized embedding! Draw a $k \times m$ Gaussian random matrix Ω and set

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

Then with probability one, $\mathbf{A} = \mathbf{E}\mathbf{F}$, where $\mathbf{E} = \mathbf{A}\mathbf{F}^\dagger$.

We do not need to know the factor \mathbf{E} ! It just never enters the computation.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p . (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix \mathbf{Z} such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

- (1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;
- (2) Form a $(k + p) \times n$ matrix \mathbf{F} holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
- (3) Do k steps of Gram-Schmidt on the columns of \mathbf{F} to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p . (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix \mathbf{Z} such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

- (1) Draw a $(k + p) \times m$ random matrix Ω ;
- (2) Form a $(k + p) \times n$ matrix \mathbf{F} holding samples from the row space, $\mathbf{F} = \Omega\mathbf{A}$;
- (3) Do k steps of Gram-Schmidt on the columns of \mathbf{F} to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Basic version: Use a *Gaussian* random matrix.

Complexity is $O(mnk)$ for a general dense matrix. Very high *practical* speed.

Complexity is $O(\text{nnz}(\mathbf{A})k)$ for a sparse matrix. Again, high *practical* speed.

In some ways optimal sampling. Well supported by theory, e.g.:

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

(Skeletonization slightly increases the error: $\|\mathbf{A} - \mathbf{A}(:, J_s)\mathbf{Z}\| \geq \|\mathbf{A} - \mathbf{A}\mathbf{F}^\dagger\mathbf{F}\|$.)

Note: The probability that a set of k columns is sampled is in a certain sense proportional to its spanning volume. This is precisely the property we are after.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p . (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix \mathbf{Z} such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

- (1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;
- (2) Form a $(k + p) \times n$ matrix \mathbf{F} holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
- (3) Do k steps of Gram-Schmidt on the columns of \mathbf{F} to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: Use a Gaussian matrix $\mathbf{\Omega}$, and incorporate *power iteration*.

Replace $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$ in step (2) by $\mathbf{F} = \mathbf{\Omega}(\mathbf{A}\mathbf{A}^*)^q\mathbf{A}$ for a small integer q . Say $q = 1$ or $q = 2$.
(I.e. do classical subspace iteration.)

This makes $\text{row}(\mathbf{F})$ better aligned with the space spanned by the dominant k right singular vectors of \mathbf{A} .

Enhances accuracy, at cost of more work. Strong supporting theory. E.g., with $p = k$:

$$\mathbb{E} \left[\|\mathbf{A} - \mathbf{A}\mathbf{F}^\dagger\mathbf{F}\| \right] \leq \left(1 + 4\sqrt{\frac{2 \min(m, n)}{k-1}} \right)^{1/(2q+1)} \sigma_{k+1}(\mathbf{A}).$$

Re-orthonormalization is sometimes required to avoid loss of accuracy due to round-off.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p . (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix \mathbf{Z} such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

- (1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;
- (2) Form a $(k + p) \times n$ matrix \mathbf{F} holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
- (3) Do k steps of Gram-Schmidt on the columns of \mathbf{F} to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: Use a *structured random matrix* $\mathbf{\Omega}$. (“Fast Johnson-Lindenstrauss transform”)

Idea: Use a matrix $\mathbf{\Omega}$ for which $\mathbf{\Omega}\mathbf{A}$ can be evaluated efficiently.

- Subsampled Randomized Fourier Transform (SRFT). Can be applied using FFT like methods. Cost is $O(mn \log(k))$ instead of $O(mnk)$.
- Sparse random matrix. Put only a couple of non-zero entries in each column. (Entries can be restricted to ± 1 for additional efficiency.) $O(mn)$ cost attainable.

Works quite well in practice. SRFTs are almost as good as Gaussians.

However, *far* weaker theory is available.

Cannot be combined with power iteration.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p . (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix \mathbf{Z} such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

- (1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;
- (2) Form a $(k + p) \times n$ matrix \mathbf{F} holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
- (3) Do k steps of Gram-Schmidt on the columns of \mathbf{F} to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: Discrete empirical interpolation method (DEIM)

Idea: Use the sample matrix \mathbf{F} to build an approximate truncated SVD of \mathbf{A} .

(Form SVD of $(\mathbf{A}\mathbf{F}^\dagger)\mathbf{F}$.) Requires one additional matrix-matrix multiplication.

Then perform partially pivoted LU on a thin matrix formed by the singular vectors to pick the spanning columns.

DEIM sometimes produces slightly more optimal column selection than CPQR.

DEIM can be faster than doing CPQR directly. (Which is perhaps counterintuitive!)

Often excellent choice.

Reference: Sorensen & Embree SISC 2016.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p . (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix \mathbf{Z} such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

- (1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;
- (2) Form a $(k + p) \times n$ matrix \mathbf{F} holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
- (3) Do k steps of Gram-Schmidt on the columns of \mathbf{F} to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: “Poor man’s DEIM”

Idea: Skip forming the SVD, and just do partially pivoted LU on \mathbf{F}^* directly!

Very economical. Works about as well as either CPQR, or regular DEIM.

Becomes particularly accurate with one step of power iteration.

Ongoing work: It appears that you can get highly accurate estimates of the residual for free (almost).

Inspired by work by Trefethen and Schreiber (SIMAX 1990) on Gaussian elimination on random matrices.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix \mathbf{A} , a target rank k , and an over-sampling parameter p . (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix \mathbf{Z} such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

- (1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;
- (2) Form a $(k + p) \times n$ matrix \mathbf{F} holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
- (3) Do k steps of Gram-Schmidt on the columns of \mathbf{F} to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: Other possibilities:

- Can use sophisticated methods à la Gu-Eisenstat RRQR in Step (3). Leads to methods that are well supported by theory. Little improvement in practice, however.
- Can use the sketch to form an SVD. Then estimate *leverage scores*, and draw the columns through randomized sampling on the index set $\{1, 2, \dots, n\}$ using the resulting probability distribution. Rarely competitive in practice.
(However, approaches of this type can be powerful for *huge* matrices where the matrix-vector multiplication is not accessible.)

Numerical experiments

Numerical experiments

Question: Which type of random matrix should I use for the sketching?

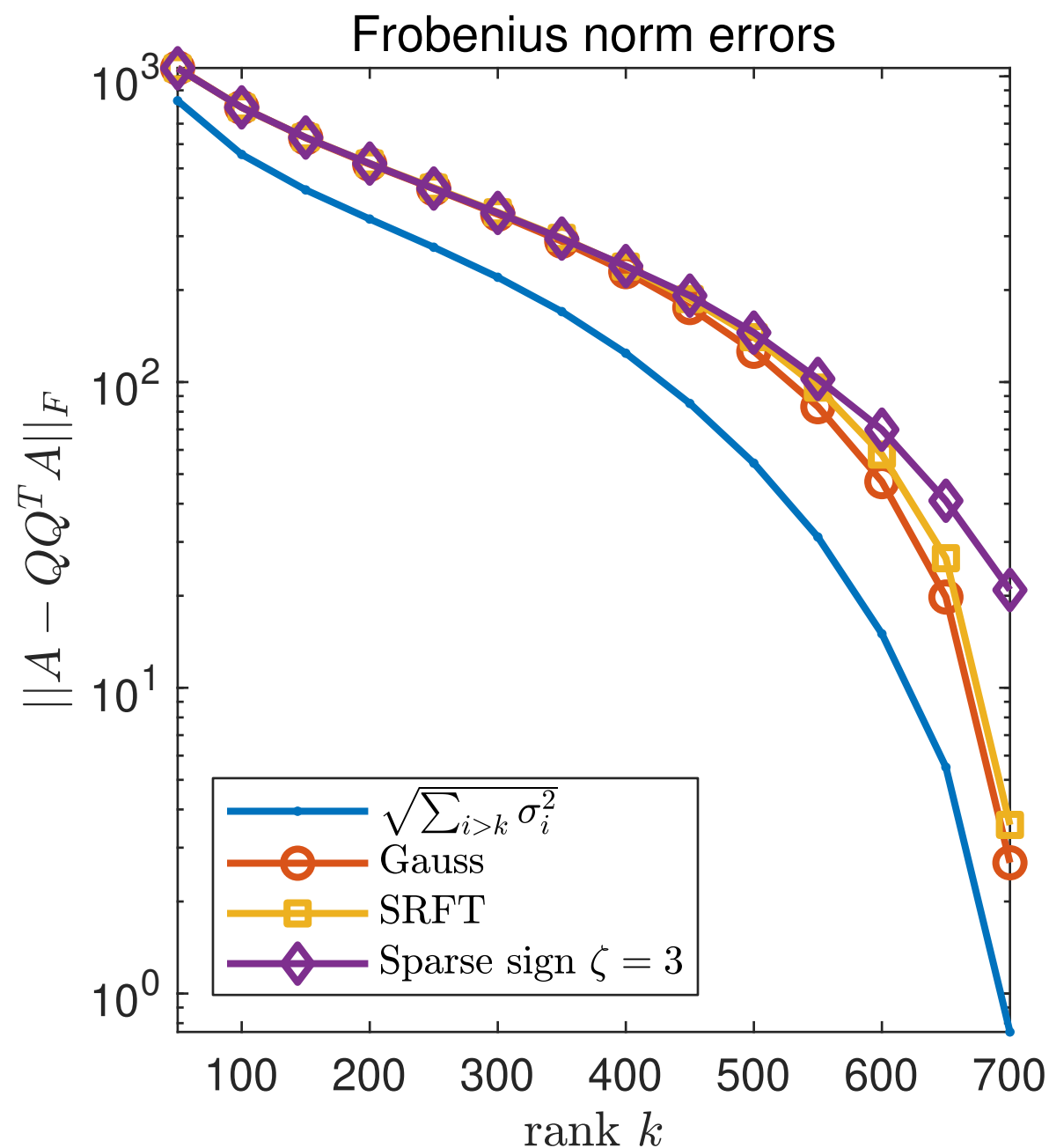
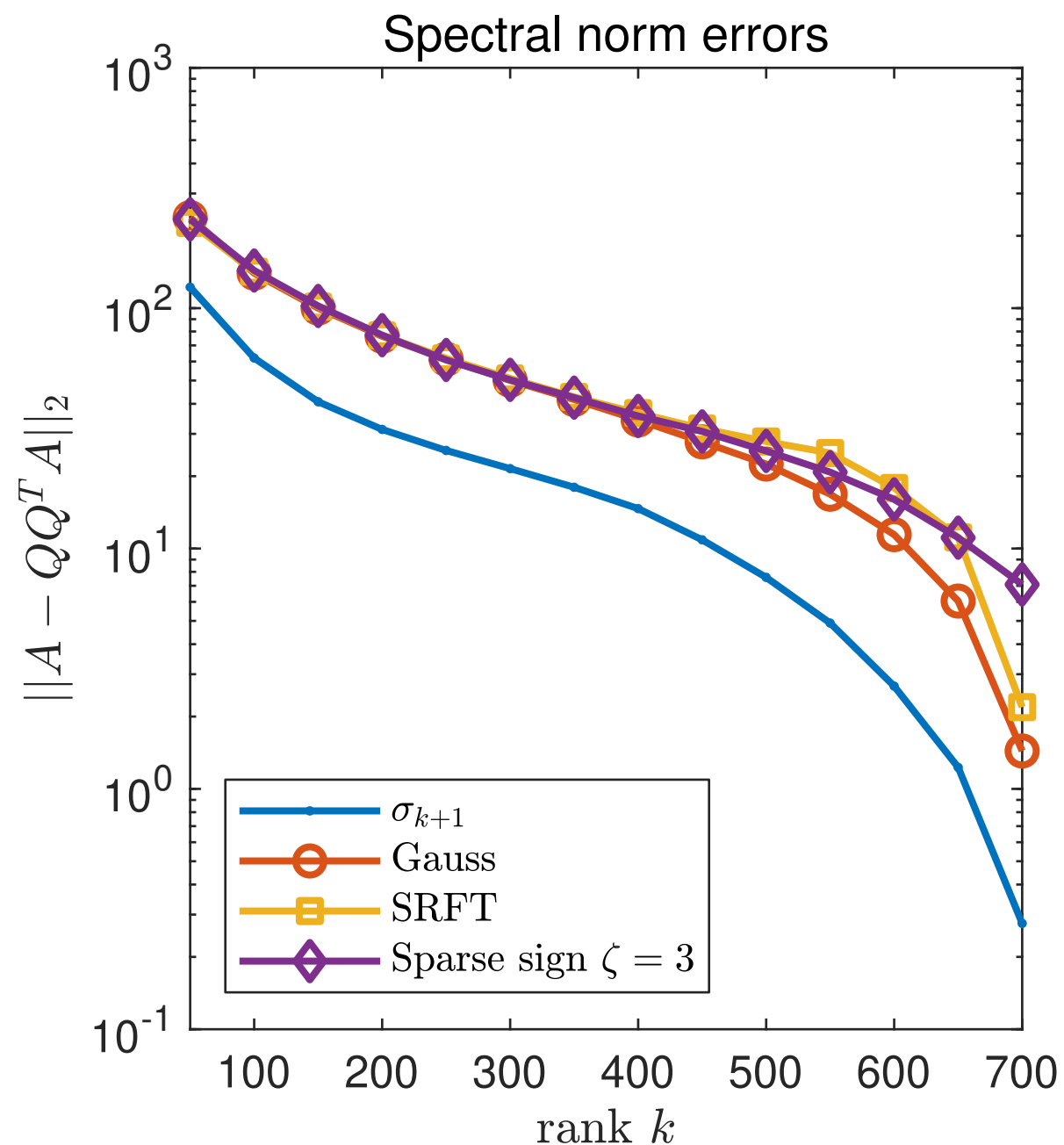
We will compare:

- Optimality: How good of a basis for the row space do you get?
- Computational cost: What is the *practical* speed?

Comparison of different random matrices — accuracy

Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

(To be precise, in these experiments, we sketched the *column space*, not the row space.)

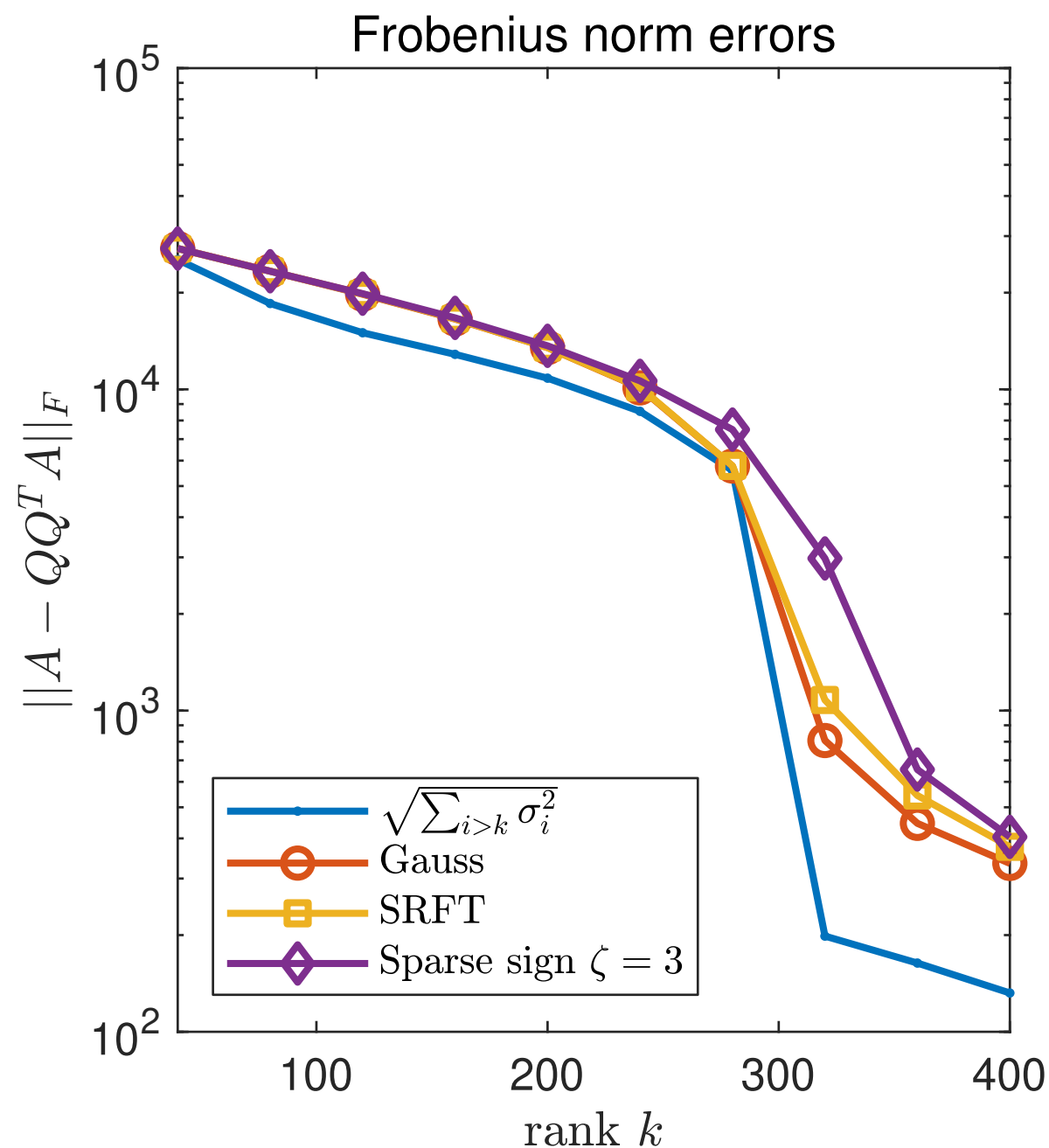
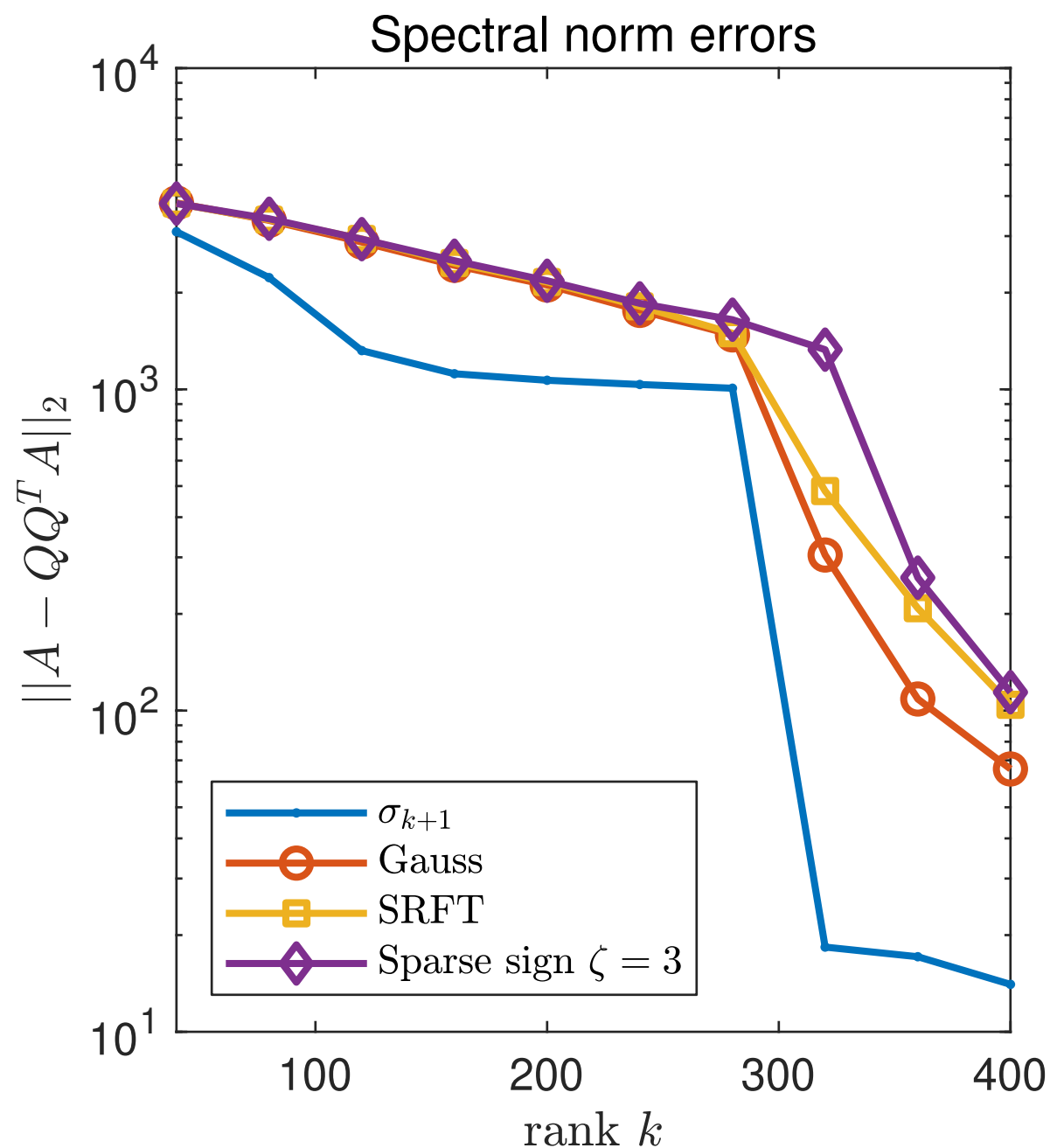


The “MNIST” test matrix is dense and of size $784 \times 60\,000$ where each column holds one hard drawn digit between 0 and 9. The matrix is 80% sparse.

Comparison of different random matrices — accuracy

Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

(To be precise, in these experiments, we sketched the *column space*, not the row space.)

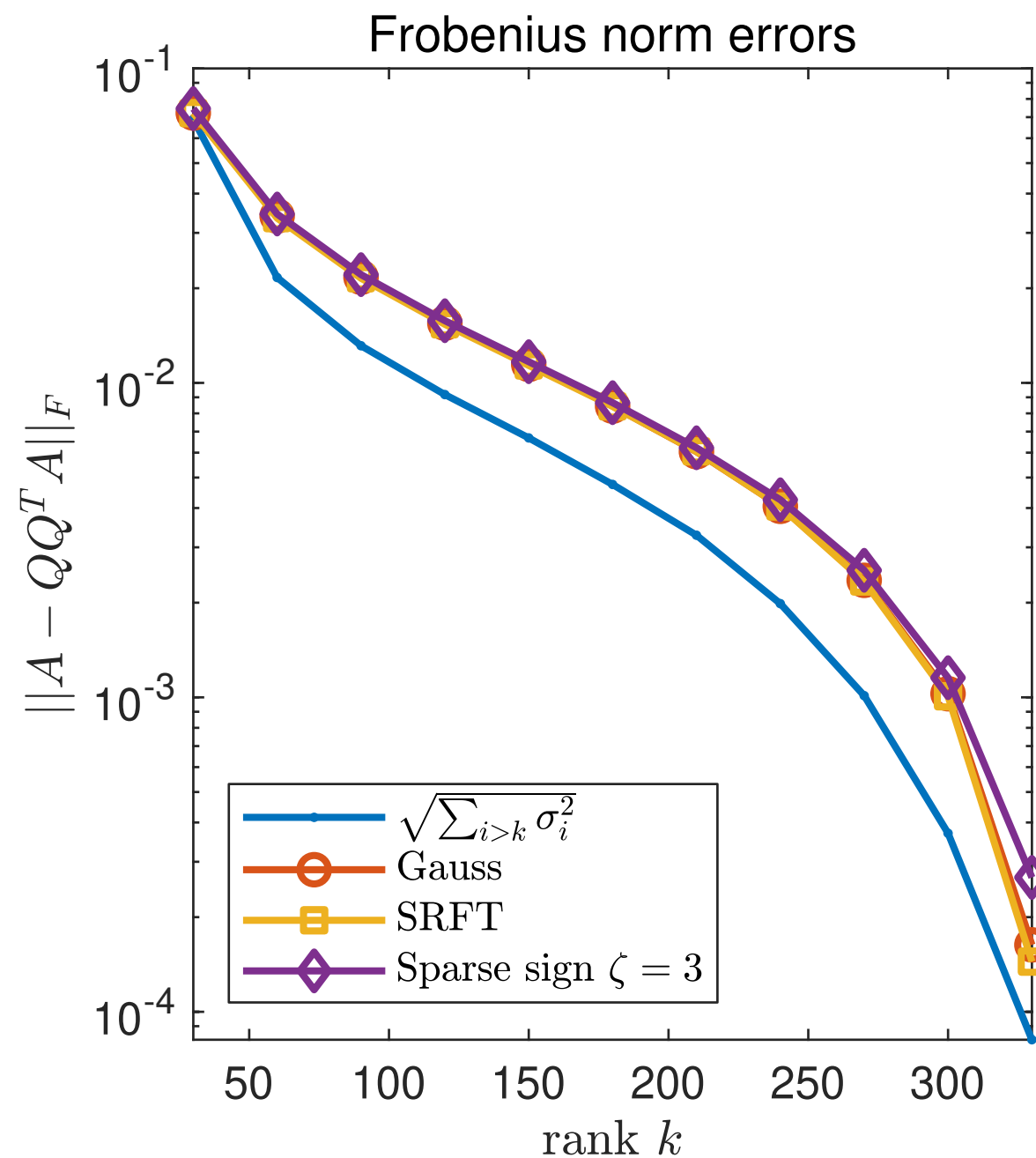
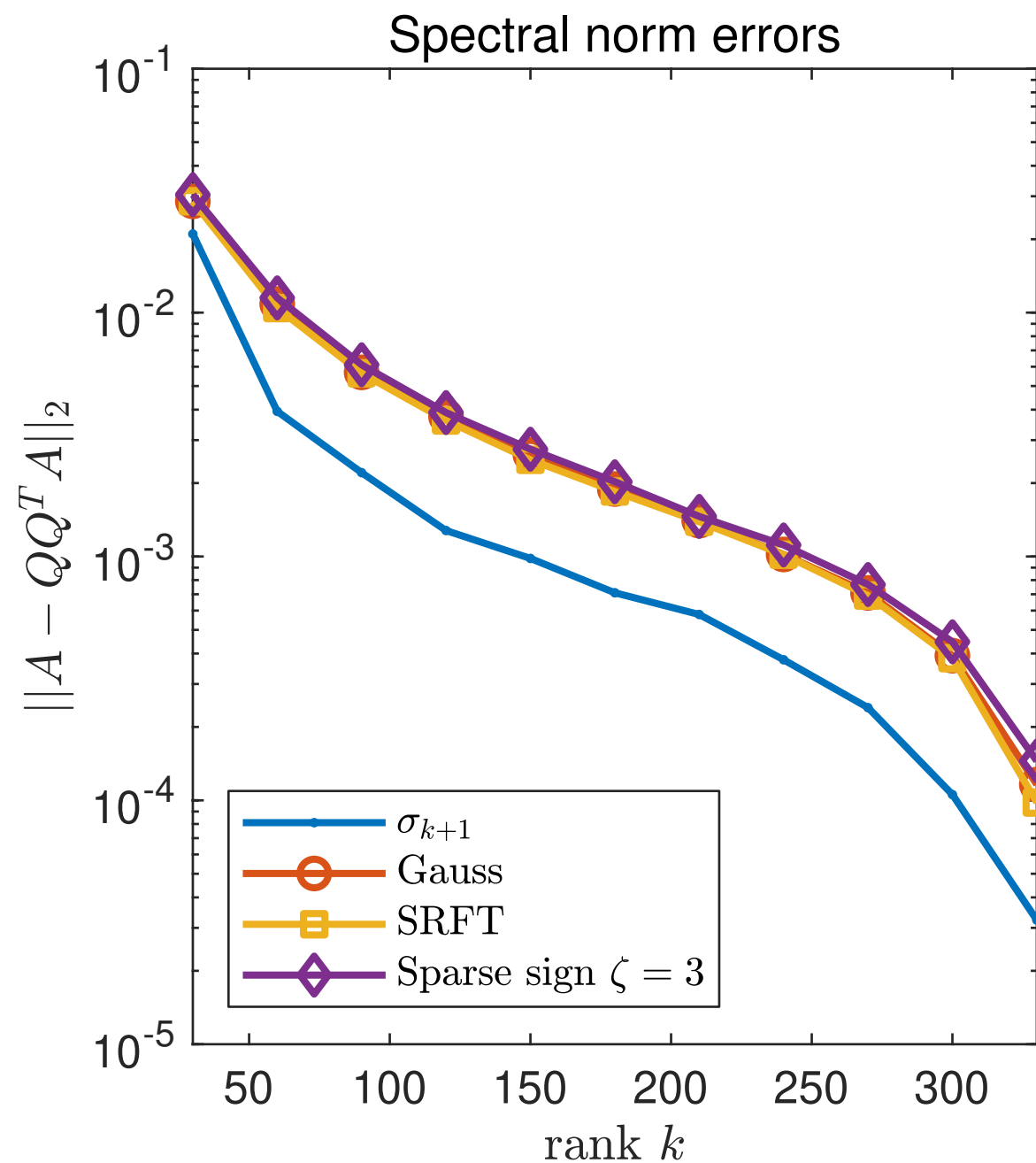


The “LARGE” test matrix is taken from a linear programming example. It is sparse, of size $4\,282 \times 8\,617$, with 20 635 nonzero entries.

Comparison of different random matrices — accuracy

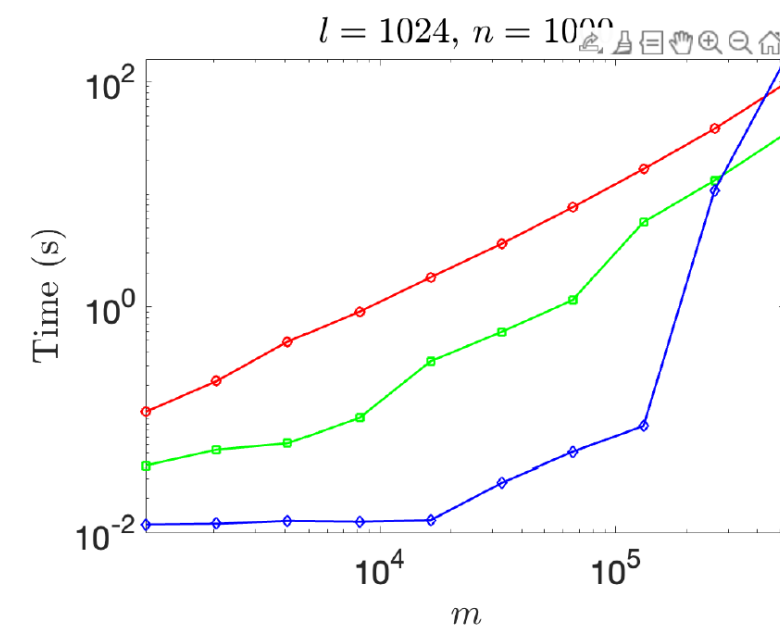
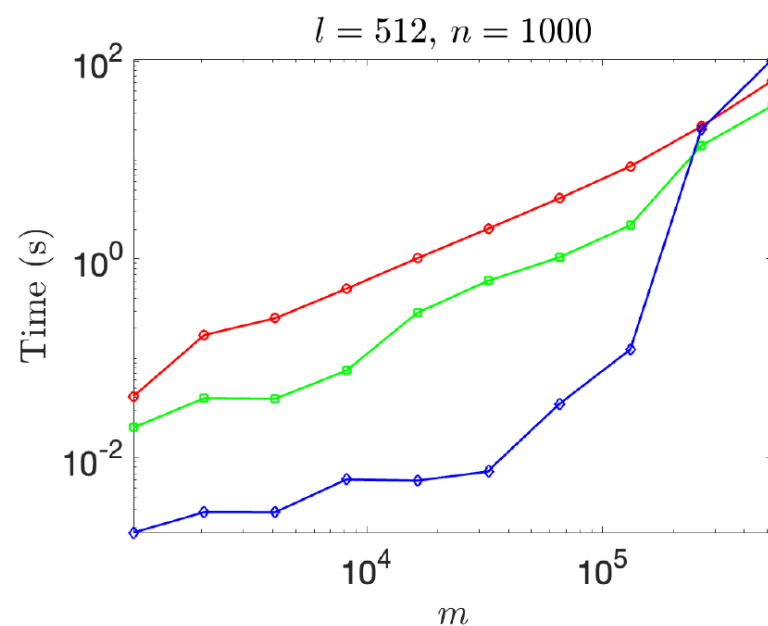
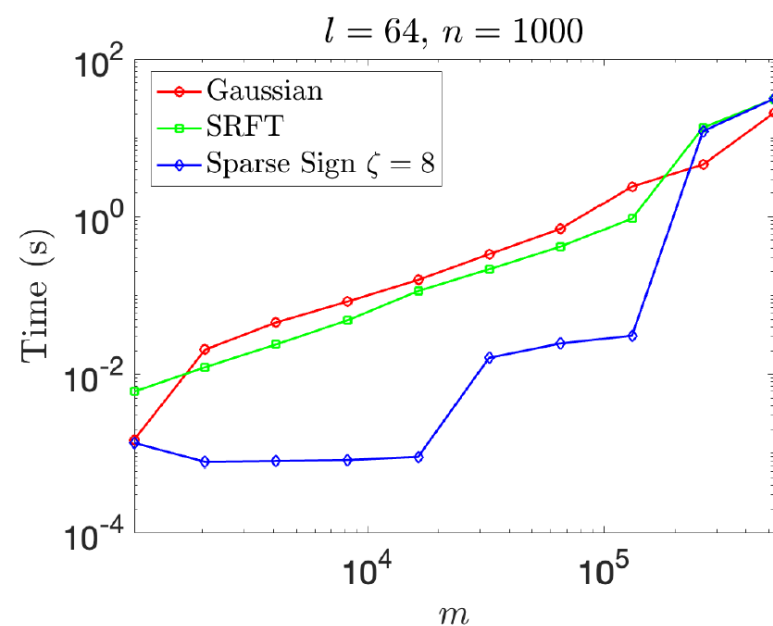
Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

(To be precise, in these experiments, we sketched the *column space*, not the row space.)



The “*snn*” test matrix has been used in the CUR literature before. It is an artificial sparse matrix of size $1\,000 \times 1\,000$.

Comparison of different random matrices — execution time



*The runtime of applying different subspace embeddings $\Omega \in \mathbb{R}^{\ell \times m}$ to an arbitrary **dense** matrix of size $m \times n$, scaled with respect to the ambient dimension m , at different embedding dimension l and a fixed number of columns $n = 1000$.*

Note: Observe that the dimension of the sketch is quite high in these examples.

Numerical experiments

Question: How should I postprocess the matrix, once I have extracted a sketch?

We will compare:

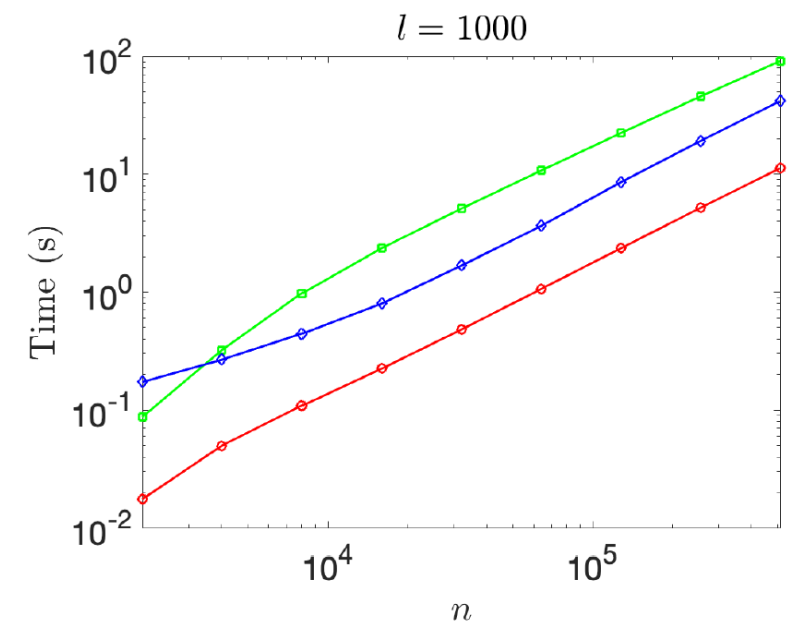
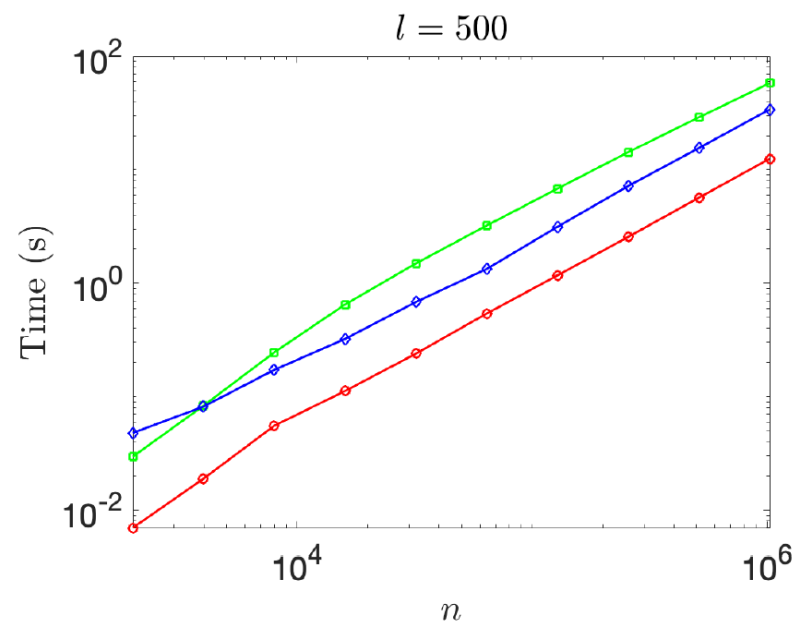
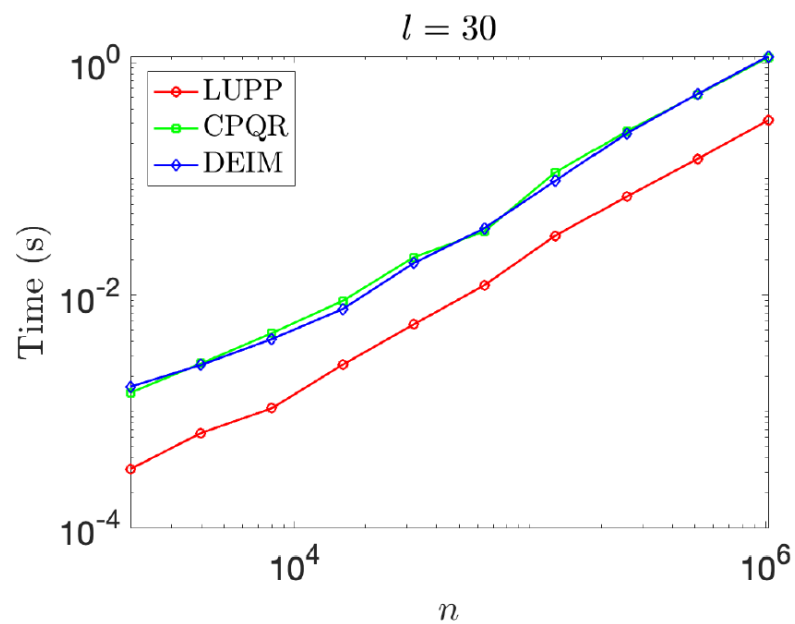
- Column pivoted QR
- DEIM: Form approximate SVD, then partially pivoted LU on the singular vectors.
- Partially pivoted LU directly on the sketching matrix. (“Poor man’s DEIM”)
- (Form approximate RSVD, then compute “leverage scores”, then draw columns based on the leverage scores.)

In these experiments, we use *Gaussian* random matrices.

Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$.

These are experiments to test the *runtime*.



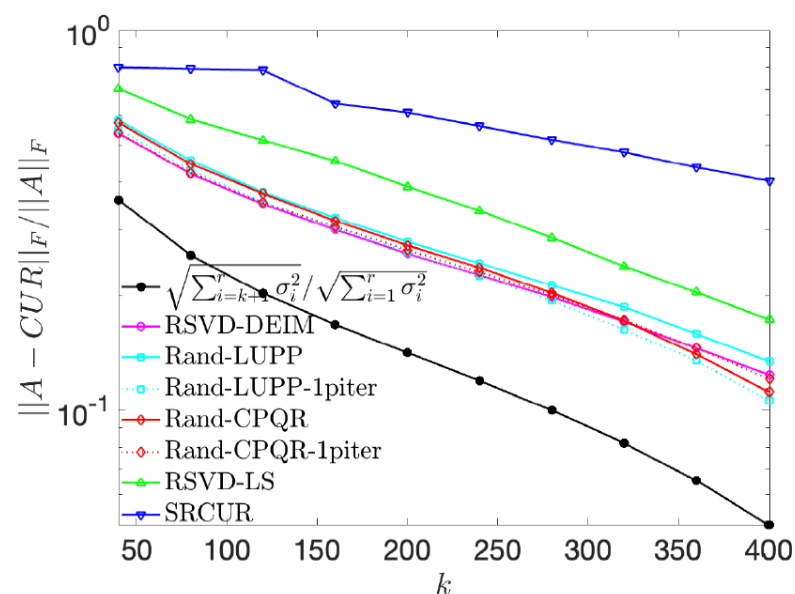
The runtime of various pivoting schemes on the sketches of size $n \times \ell$, scaled with respect to the problem size n , at different embedding dimension ℓ .

Observe that the dimension of the sketch is quite high in these examples.

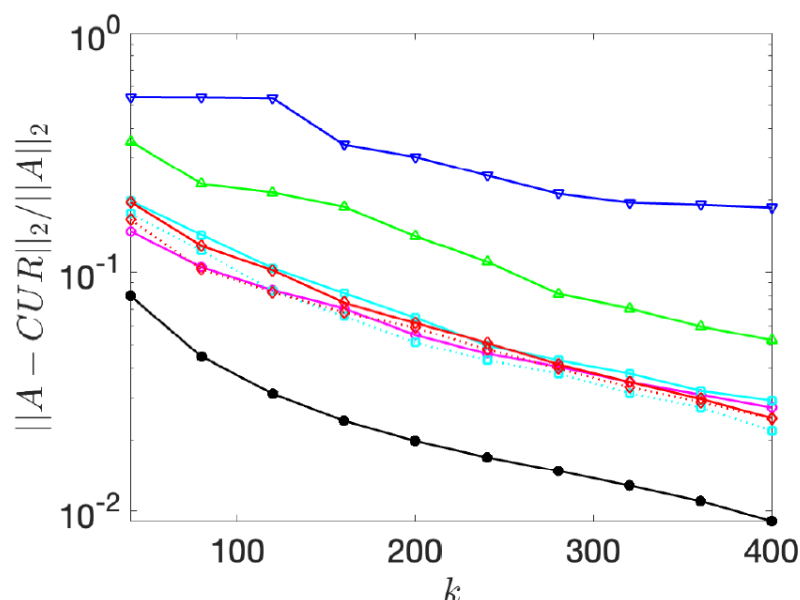
Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$.

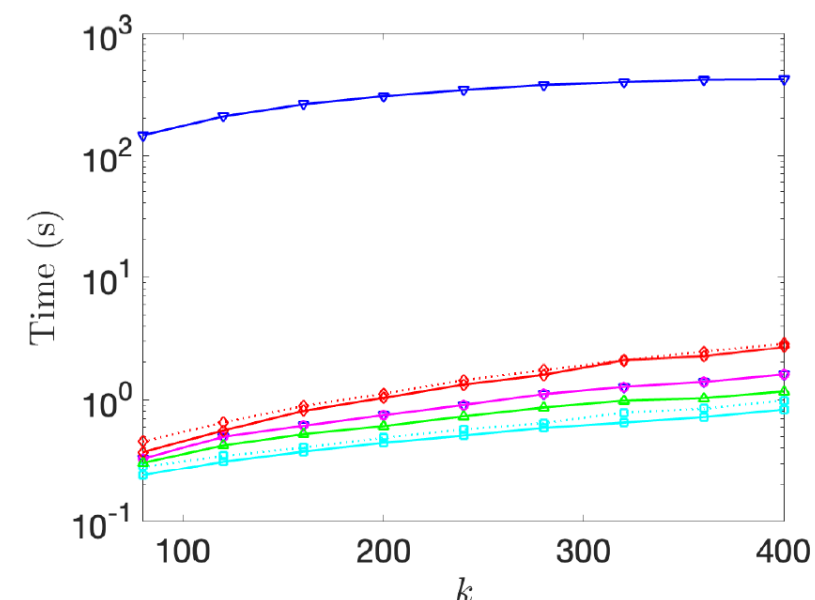
These are experiments to test the *accuracy / optimality*.



(a) Frobenius norm error.



(b) Spectral norm error.



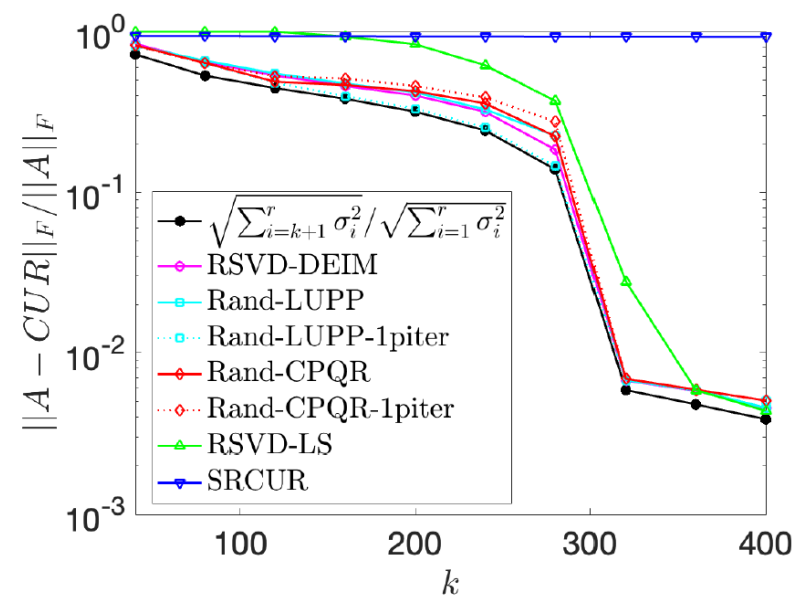
(c) Runtime.

The “MNIST” test matrix is dense and of size $784 \times 60\,000$ where each column holds one hand drawn digit between 0 and 9. The matrix is 80% sparse.

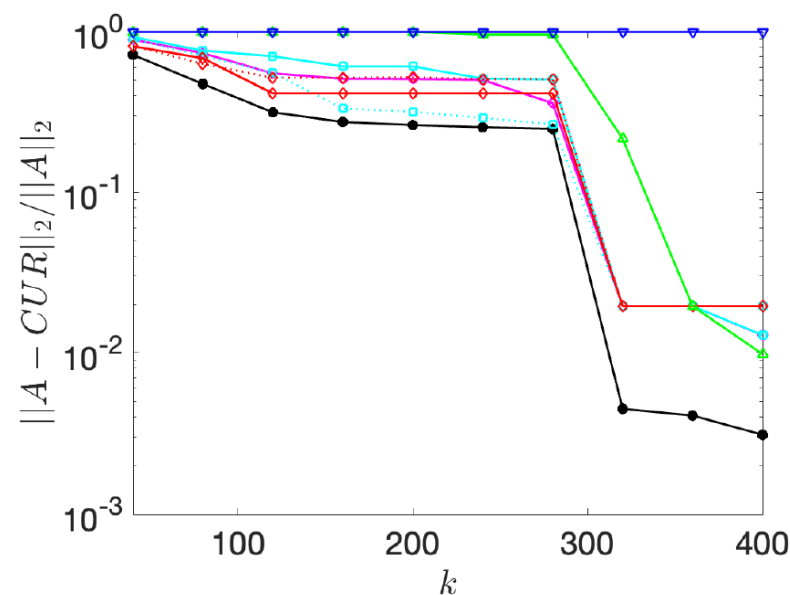
Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$.

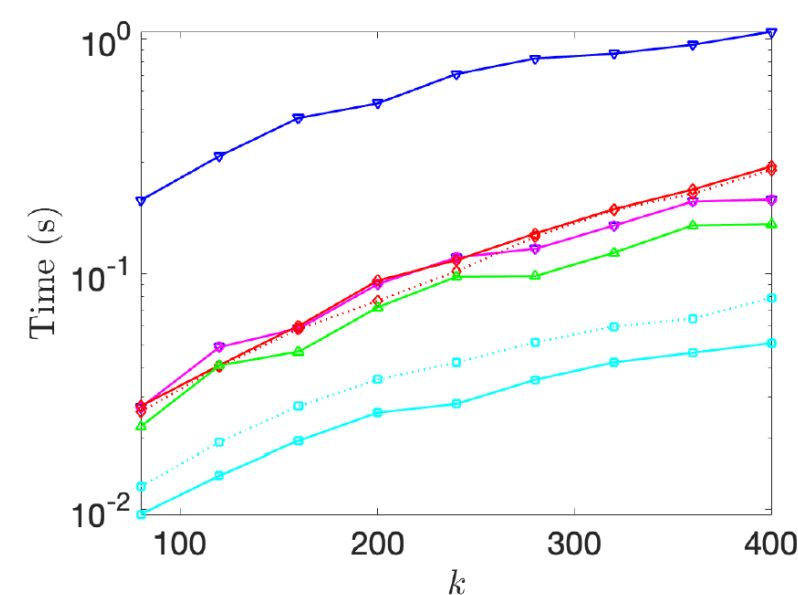
These are experiments to test the *accuracy / optimality*.



(a) Frobenius norm error.



(b) Spectral norm error.



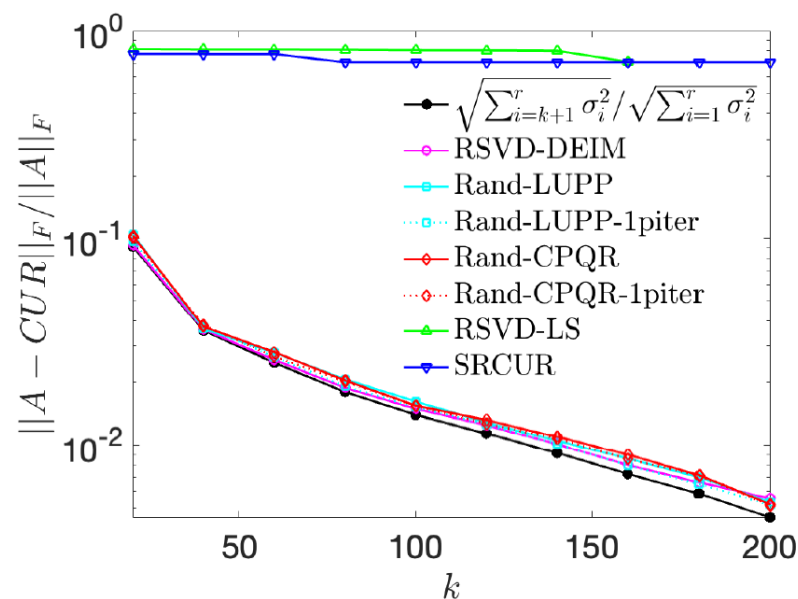
(c) Runtime.

The “YALEFACE64x64” test matrix holds 165 face images, each with 64×64 pixels. The pictures have been normalized, to create a dense matrix of size 165×4096 .

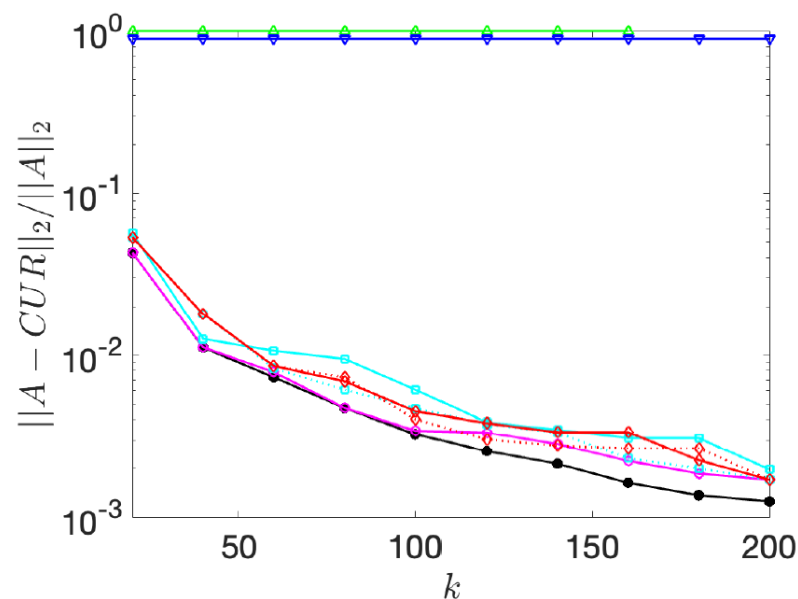
Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$.

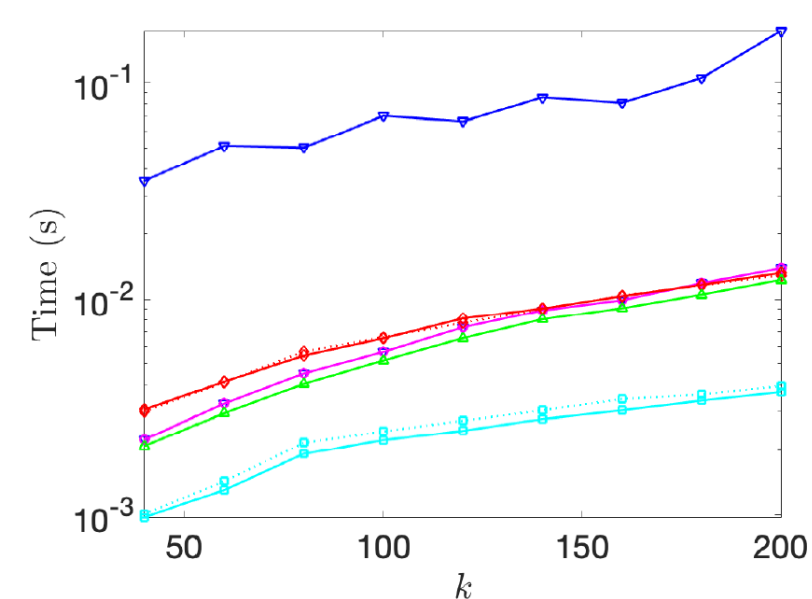
These are experiments to test the *accuracy / optimality*.



(a) Frobenius norm error.



(b) Spectral norm error.



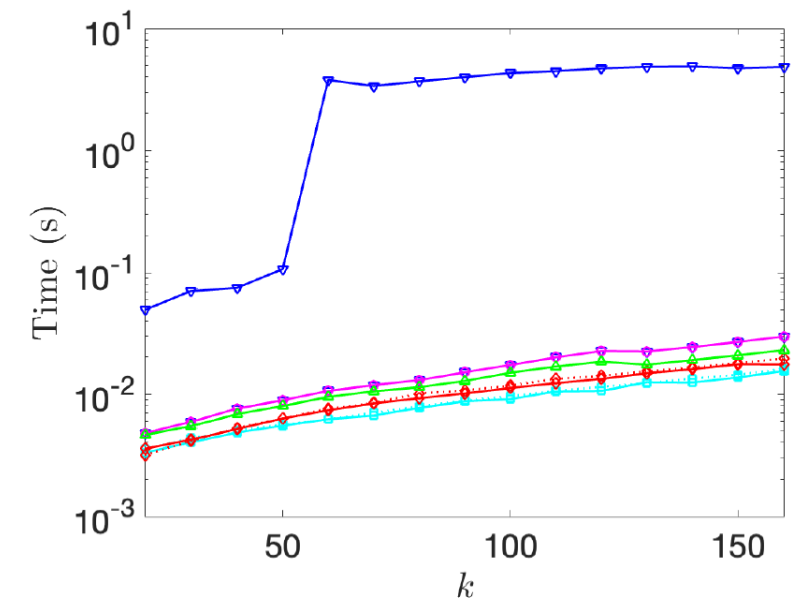
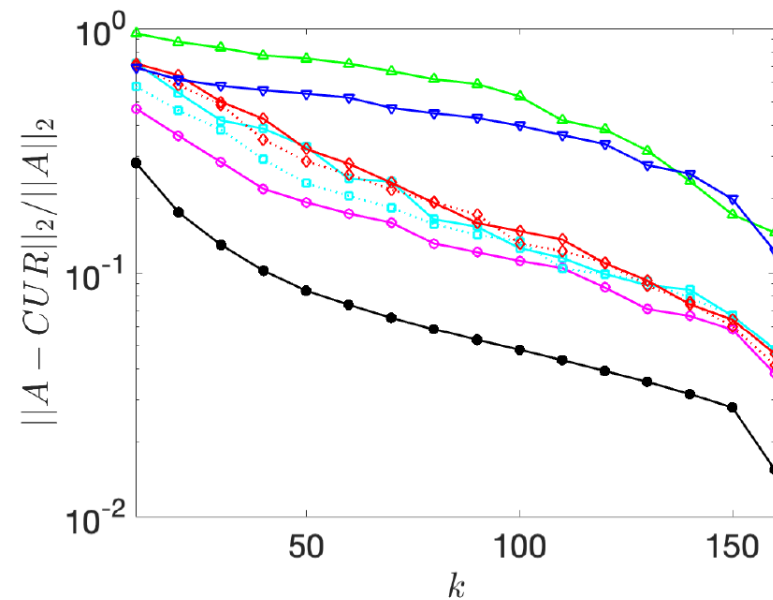
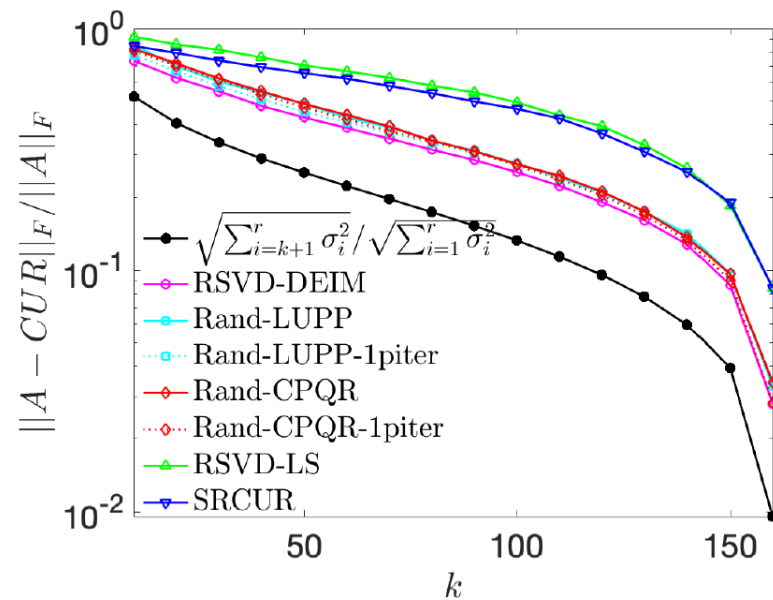
(c) Runtime.

The “sNN” test matrix has been used in the CUR literature before. It is an artificial sparse matrix of size $1\,000 \times 1\,000$.

Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$.

These are experiments to test the *accuracy / optimality*.



The “LARGE” test matrix is taken from a linear programming example. It is sparse, of size $4\,282 \times 8\,617$, with 20 635 nonzero entries.

Key points:

- Gaussian matrices are highly recommended. Excellent general purpose tools.
- “Sparse random” is *very fast* in all environments. Slightly less accurate. (“s-hashing matrix” in Cora Cartis’ talk.)
- Subsampled trigonometric transforms are about as accurate as Gaussians. When the rank is large (say 500 or 1000), you see substantial speed gain.
- We tested three methods for picking columns from the sketch matrix:
 1. Column pivoted QR.
 2. DEIM. (Compute approximate RSVD, then do LU with partial pivoting.)
 3. Partially pivoted LU.

They are about equally good at picking columns. DEIM perhaps slight winner.

Partially pivoted LU (“Poor man’s DEIM”) is the fastest by a margin.

More detailed analysis in Anil Damle’s talk yesterday.

Key points:

- Gaussian matrices are highly recommended. Excellent general purpose tools.
- “Sparse random” is *very fast* in all environments. Slightly less accurate. (“s-hashing matrix” in Cora Cartis’ talk.)
- Subsampled trigonometric transforms are about as accurate as Gaussians. When the rank is large (say 500 or 1000), you see substantial speed gain.
- We tested three methods for picking columns from the sketch matrix:
 1. Column pivoted QR.
 2. DEIM. (Compute approximate RSVD, then do LU with partial pivoting.)
 3. Partially pivoted LU.

They are about equally good at picking columns. DEIM perhaps slight winner.

Partially pivoted LU (“Poor man’s DEIM”) is the fastest by a margin.

Fun bonus: The idea of picking pivot columns via randomized sketching can also be used to solve a long-standing open problem of how to *block* column-pivoted QR. In other words, you want to move the vast majority of the flops from BLAS2 to BLAS3 operations.

Randomized pivoting in Householder QR

Given a dense $n \times n$ matrix \mathbf{A} , compute a column pivoted QR factorization

$$\begin{array}{ccccccc} \mathbf{A} & \mathbf{P} & \approx & \mathbf{Q} & \mathbf{R}, \\ n \times n & n \times n & & n \times n & n \times n \end{array}$$

where, as usual, \mathbf{Q} should be ON, \mathbf{P} is a permutation, and \mathbf{R} is upper triangular.

Randomized pivoting in Householder QR

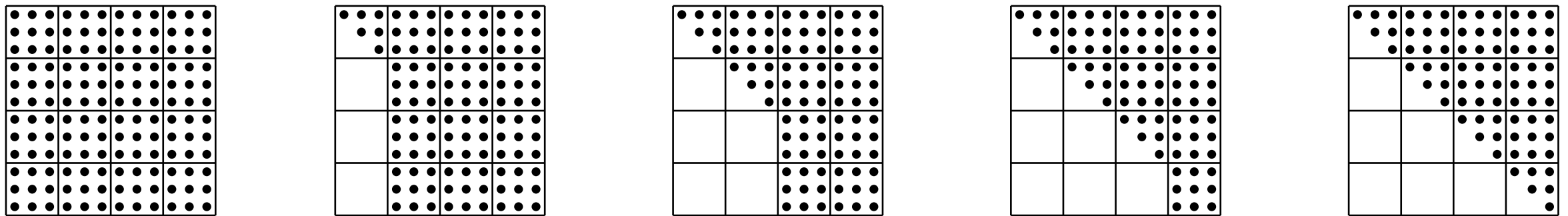
Given a dense $n \times n$ matrix \mathbf{A} , compute a column pivoted QR factorization

$$\mathbf{A} \mathbf{P} \approx \mathbf{Q} \mathbf{R},$$

$$n \times n \quad n \times n \quad n \times n \quad n \times n$$

where, as usual, \mathbf{Q} should be ON, \mathbf{P} is a permutation, and \mathbf{R} is upper triangular.

The technique proposed is based on a *blocked* version of classical Householder QR:



$$\mathbf{A}_0 = \mathbf{A} \quad \mathbf{A}_1 = \mathbf{Q}_1^* \mathbf{A}_0 \mathbf{P}_1 \quad \mathbf{A}_2 = \mathbf{Q}_2^* \mathbf{A}_1 \mathbf{P}_2 \quad \mathbf{A}_3 = \mathbf{Q}_3^* \mathbf{A}_2 \mathbf{P}_3 \quad \mathbf{A}_4 = \mathbf{Q}_4^* \mathbf{A}_3 \mathbf{P}_4$$

Each \mathbf{P}_j is a permutation matrix computed via randomized sampling.

Each \mathbf{Q}_j is a product of Householder reflectors.

The key challenge has been to find good permutation matrices.

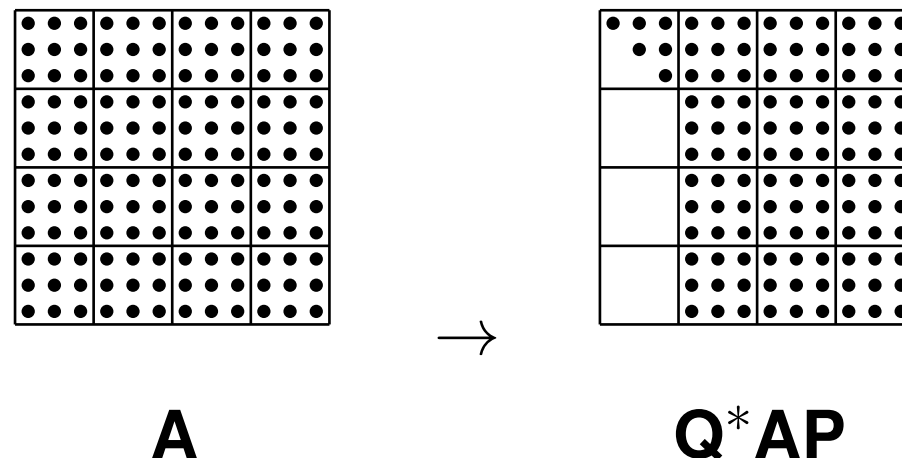
We seek \mathbf{P}_j so that the set of b chosen columns *has maximal spanning volume*.

The pivot selection problem is *very* closely related to the problem of finding spanning columns that we started with! The likelihood that any block of columns is “hit” by the random vectors is directly proportional to its volume. Perfect optimality is *not* required.

Randomized pivoting in Householder QR

How to do block pivoting using randomization:

Let \mathbf{A} be of size $m \times n$, and let b be a block size.



\mathbf{Q} is a product of b Householder reflectors.

\mathbf{P} is a permutation matrix that moves b “pivot” columns to the leftmost slots.

We seek \mathbf{P} so that the set of chosen columns *has maximal spanning volume*.

Draw a Gaussian random matrix \mathbf{G} of size $b \times m$ and form

$$\mathbf{F} = \mathbf{G} \mathbf{A}$$

$$b \times n \quad b \times m \quad m \times n$$

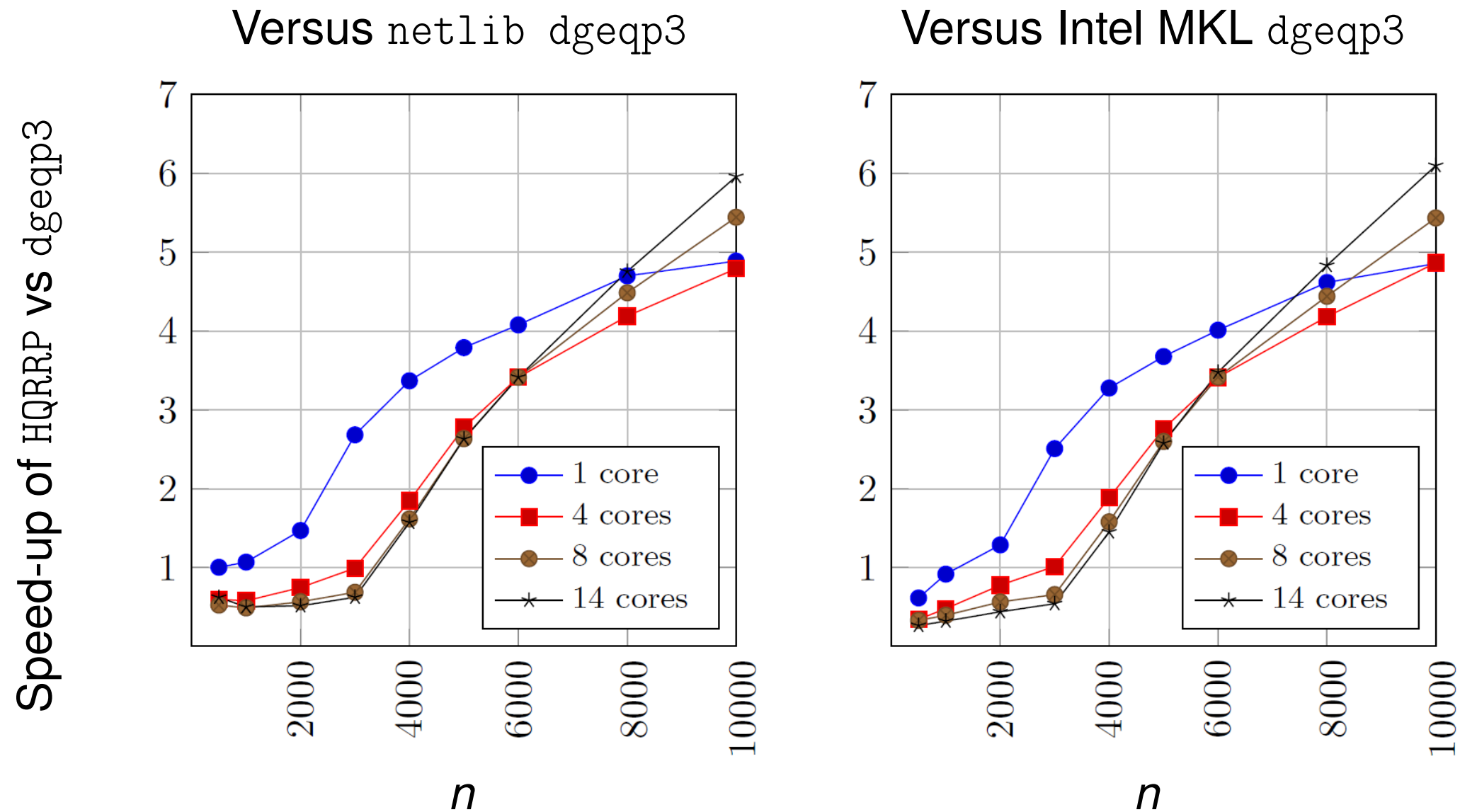
The rows of \mathbf{F} are random linear combinations of the rows of \mathbf{A} .

Then compute the pivot matrix \mathbf{P} for the first block by executing traditional column pivoting on the small matrix \mathbf{F} :

$$\mathbf{F} \mathbf{P} = \mathbf{Q}_{\text{trash}} \mathbf{R}_{\text{trash}}$$

$$b \times n \quad n \times n \quad b \times b \quad b \times n$$

Randomized pivoting in Householder QR



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

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

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

Outline of talk

- The interpolatory and CUR decompositions — what are they?
- Applications of IDs in solving PDEs and integral equations.
- Efficient algorithms for computing an interpolatory decomposition.
- PDE applications revisited: How to compress a global operator

Compression of global operators

We consider the problem of compressing certain global operators that arise in scientific computing, such as discretized integral equations. Recall for instance the BIE

$$(6) \quad q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma,$$

where Γ is a contour in \mathbb{R}^2 or a surface in \mathbb{R}^3 . The problem involves compressing *many* off-diagonal blocks in some hierarchical representation.

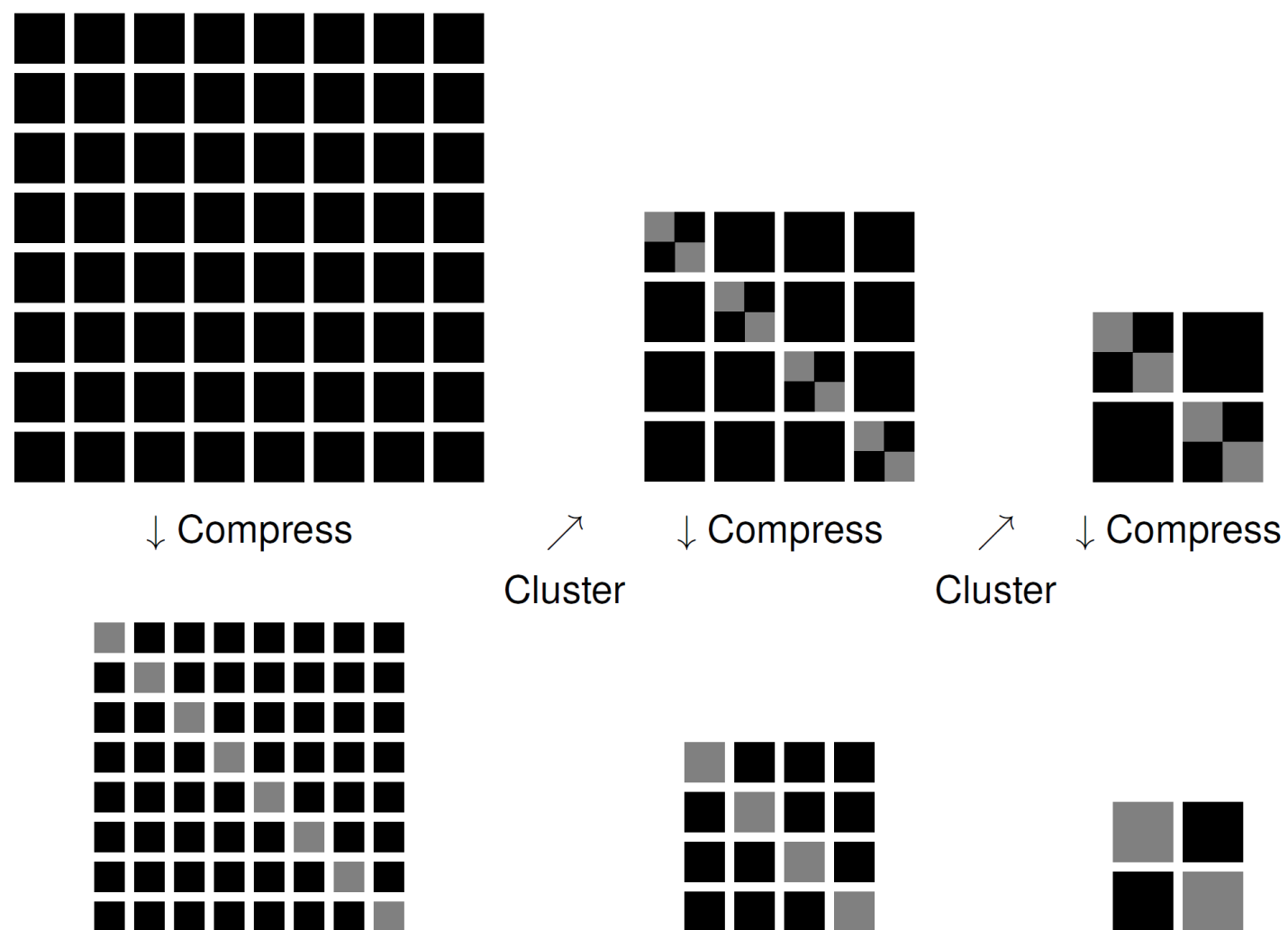
Compression of global operators

We consider the problem of compressing certain global operators that arise in scientific computing, such as discretized integral equations. Recall for instance the BIE

$$(6) \quad q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma,$$

where Γ is a contour in \mathbb{R}^2 or a surface in \mathbb{R}^3 . The problem involves compressing *many* off-diagonal blocks in some hierarchical representation.

Example from earlier — “recursive skeletonization”:



Compression of global operators

We consider the problem of compressing certain global operators that arise in scientific computing, such as discretized integral equations. Recall for instance the BIE

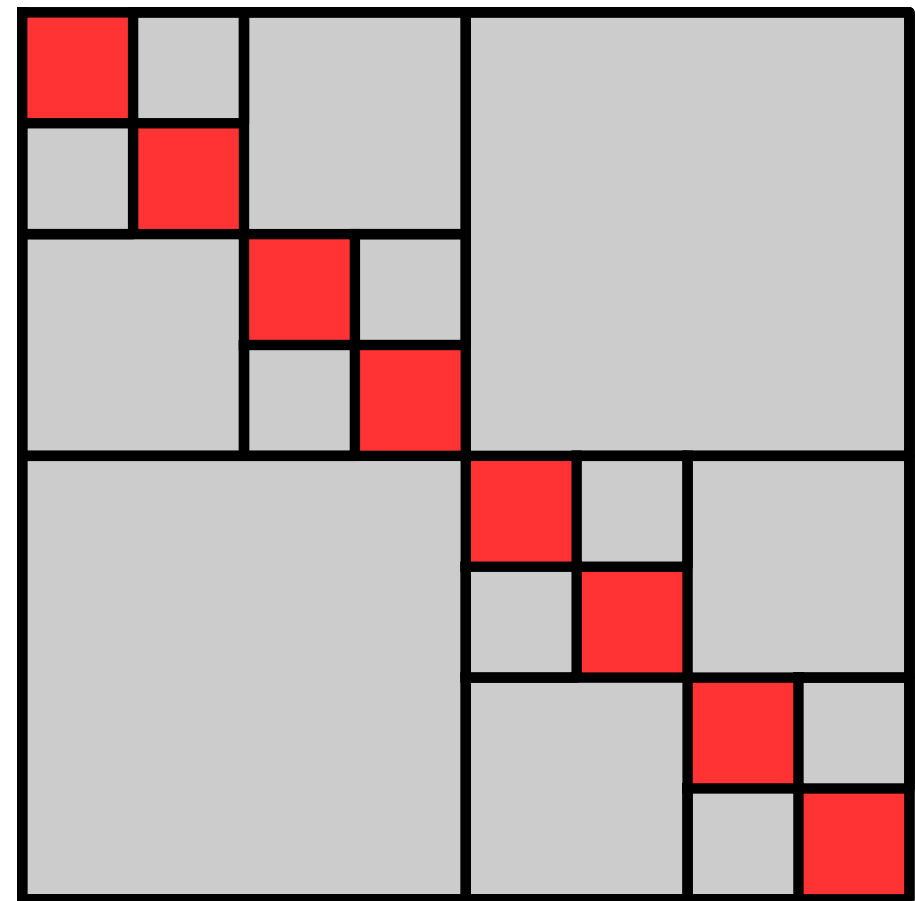
$$(6) \quad q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) ds(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma,$$

where Γ is a contour in \mathbb{R}^2 or a surface in \mathbb{R}^3 . The problem involves compressing *many* off-diagonal blocks in some hierarchical representation.

More general class of “rank structured” 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.

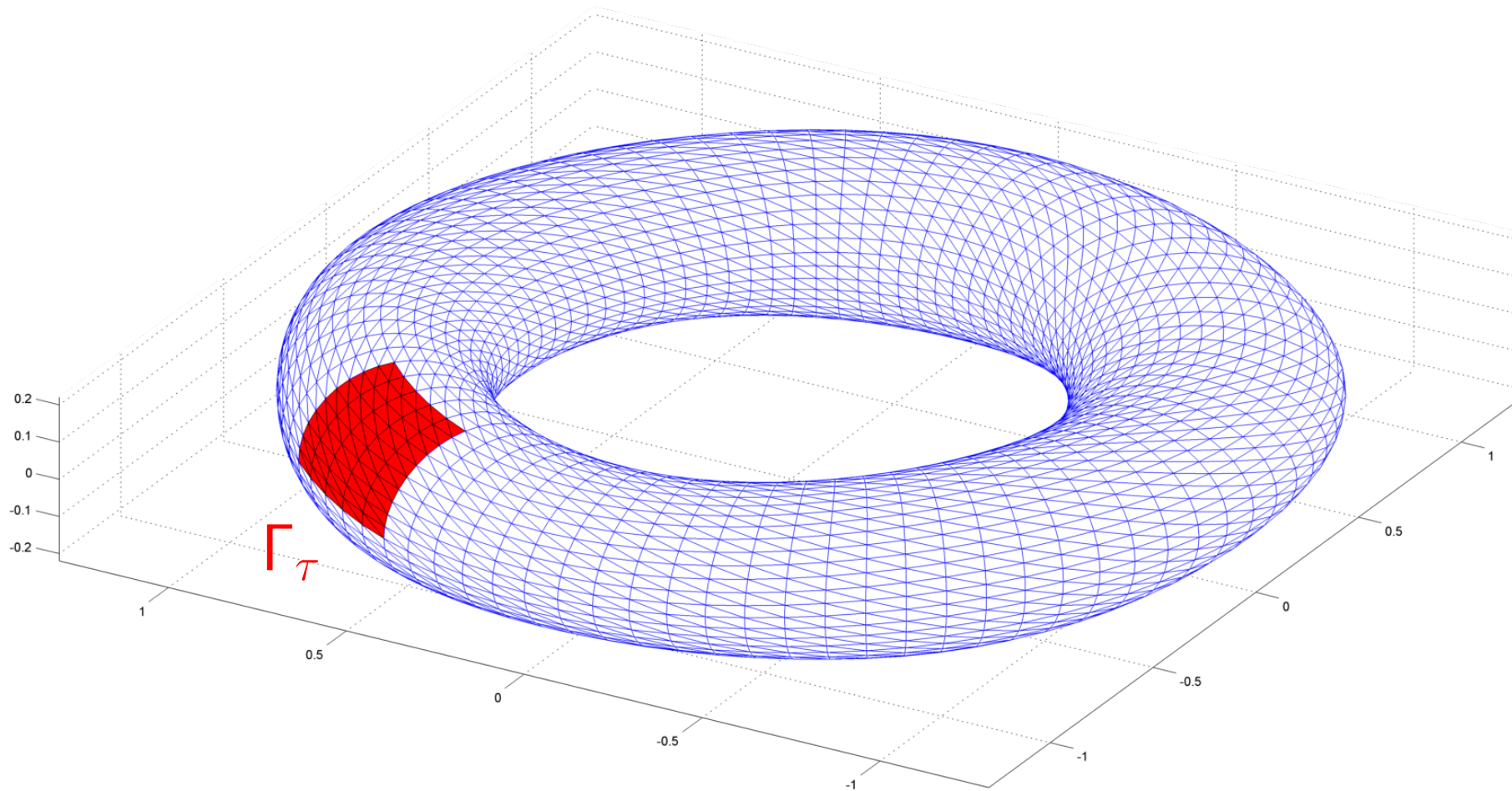
Such matrices admit linear or almost linear complexity algorithms for not only matvec, but also LU, inversion, etc.



All gray blocks have low rank.

Compression of global operators

To illustrate the problem, let \mathbf{A} denote the dense $N \times N$ matrix arising from discretizing a boundary integral equation on a contour or surface, and let Γ_τ denote a subdomain corresponding to a diagonal block $\mathbf{A}(I_\tau, I_\tau)$ in the rank structured matrix.



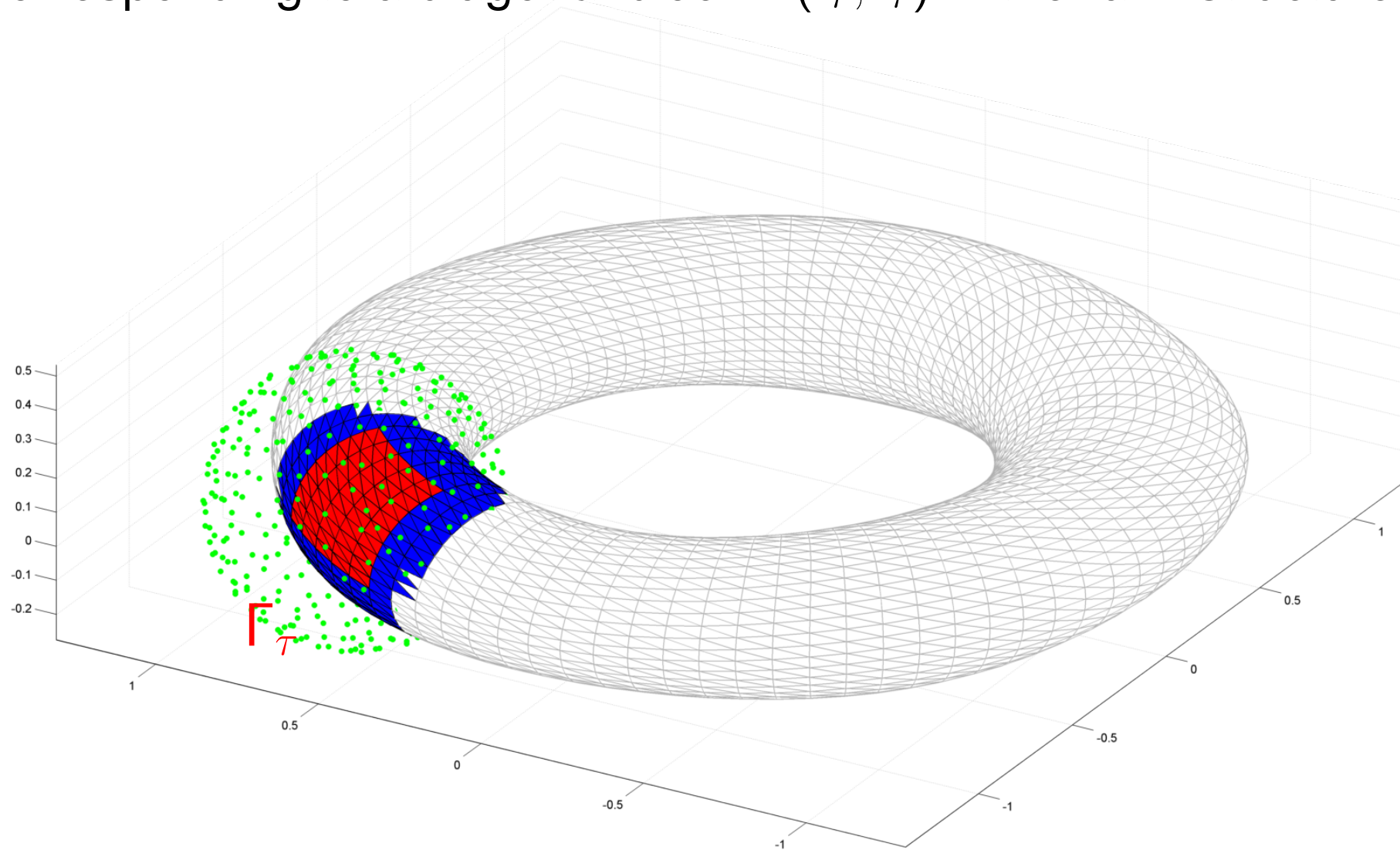
- Points in I_τ .
- Points in I_τ^c .

In recursive skeletonization, we *in principle* need to compute an ID of the matrix $\mathbf{A}(I_\tau, I_\tau^c)$.

Problem: $\mathbf{A}(I_\tau, I_\tau^c)$ can be very large! Say $10^3 \times 10^7$. And there are many such matrices!

Compression of global operators

To illustrate the problem, let \mathbf{A} denote the dense $N \times N$ matrix arising from discretizing a boundary integral equation on a contour or surface, and let Γ_τ denote a subdomain corresponding to a diagonal block $\mathbf{A}(I_\tau, I_\tau)$ in the rank structured matrix.

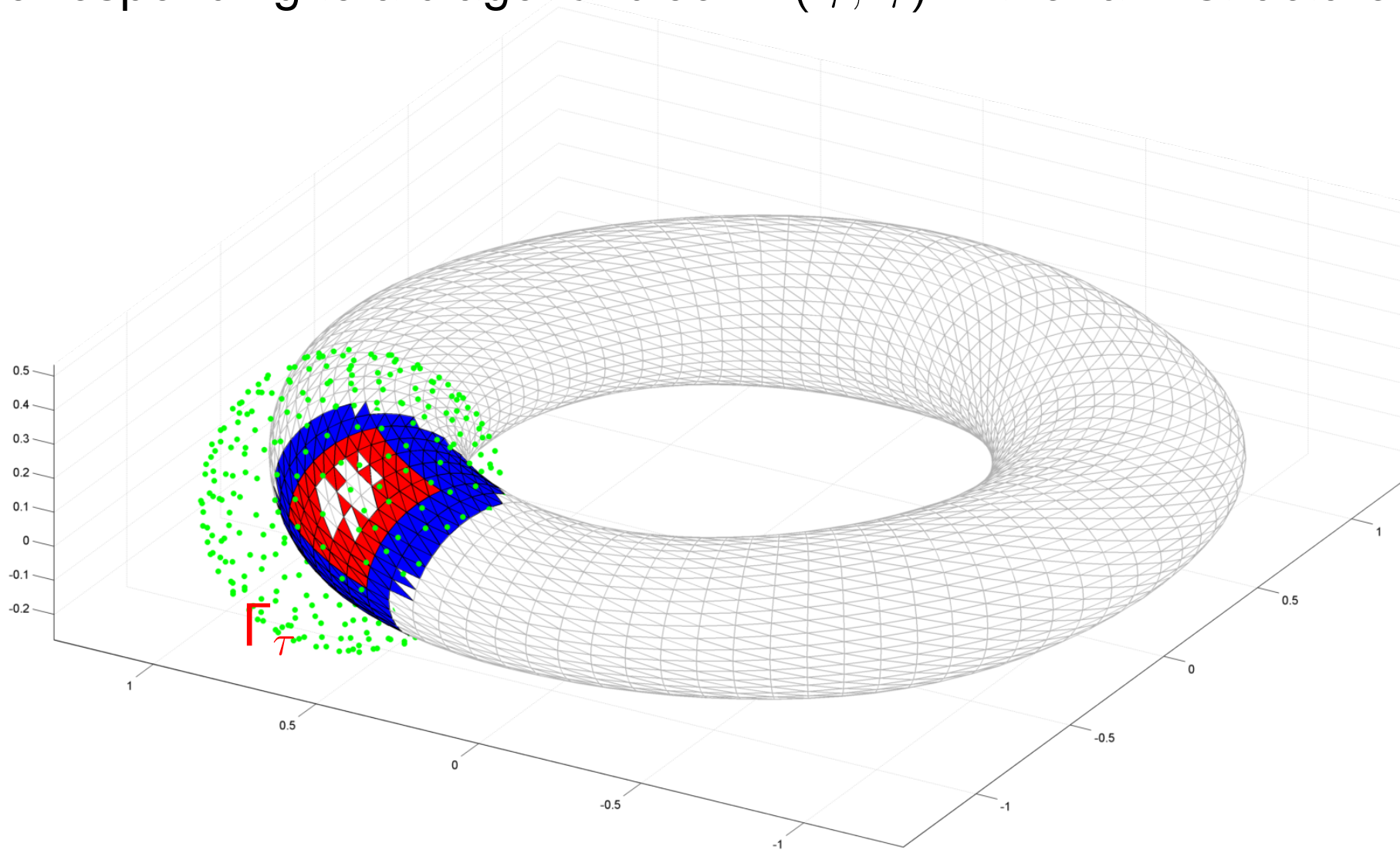


- Points in I_τ .
- Points in $I_\tau^{(\text{near})}$.
- Points in Γ_{proxy} .

One solution — the “proxy surface” method: Replace all “far field” interactions by the artificial surface Γ_{proxy} . We then only need to compute the ID of the matrix $[\mathbf{A}(I_\tau, I_\tau^{(\text{near})}) \mathbf{F}]$, where \mathbf{F} is the matrix of interaction with the proxy surface (green).

Compression of global operators

To illustrate the problem, let \mathbf{A} denote the dense $N \times N$ matrix arising from discretizing a boundary integral equation on a contour or surface, and let Γ_τ denote a subdomain corresponding to a diagonal block $\mathbf{A}(I_\tau, I_\tau)$ in the rank structured matrix.



- Points in \tilde{I}_τ .
- Points in $I_\tau^{(\text{near})}$.
- Points in Γ_{proxy} .

One solution — the “proxy surface” method: Replace all “far field” interactions by the artificial surface Γ_{proxy} . We then only need to compute the ID of the matrix $[\mathbf{A}(I_\tau, I_\tau^{(\text{near})}) \mathbf{F}]$, where \mathbf{F} is the matrix of interaction with the proxy surface (green).

Compression of global operators

The proxy surface method works well and is useful whenever you seek to compress a BIE associated with one of the standard PDEs of mathematical physics (Laplace, Helmholtz, time-harmonic Maxwell, etc.). *Highly recommended when applicable!*

Compression of global operators

The proxy surface method works well and is useful whenever you seek to compress a BIE associated with one of the standard PDEs of mathematical physics (Laplace, Helmholtz, time-harmonic Maxwell, etc.). *Highly recommended when applicable!*

We next describe a randomized algorithm for computing a data sparse representation of a global operator that can be accessed only through its action on vectors.

- In many cases, global operators are accessed by solving an associated PDE. For instance, Dirichlet-to-Neumann operators involving non-homogenous PDEs.
- Multiplication of operators, possibly involving inverses that are applied using iterative solvers. For instance, Dirichlet-to-Neumann operators involving homogenous PDEs. Or multi-physics problems.
- Compression of Schur complements that arise in the LU or Cholesky factorization of sparse matrices.

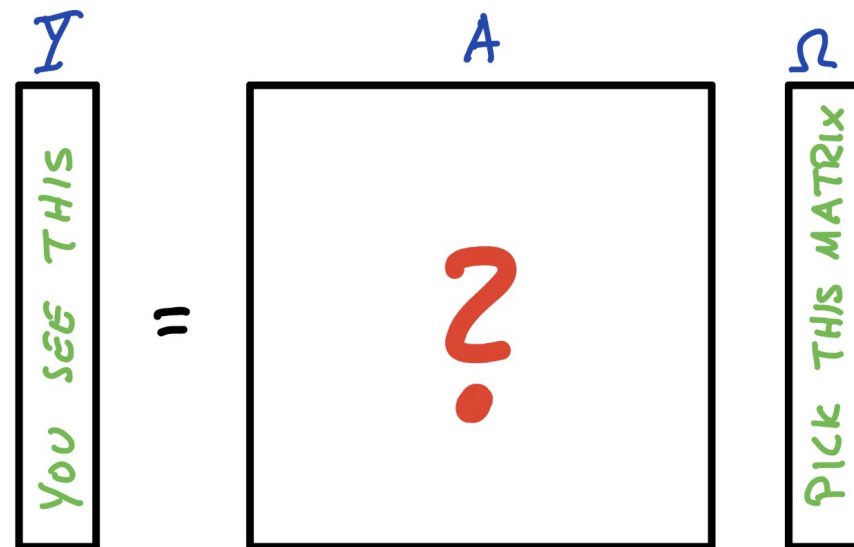
Also, even in cases where you in principle *could* apply the proxy surface method, you may not want to go to the trouble of writing a new code if you already have access to a legacy code (e.g. the Fast Multipole Method) for the matrix-vector multiplication.

Compression of global operators – problem definition:

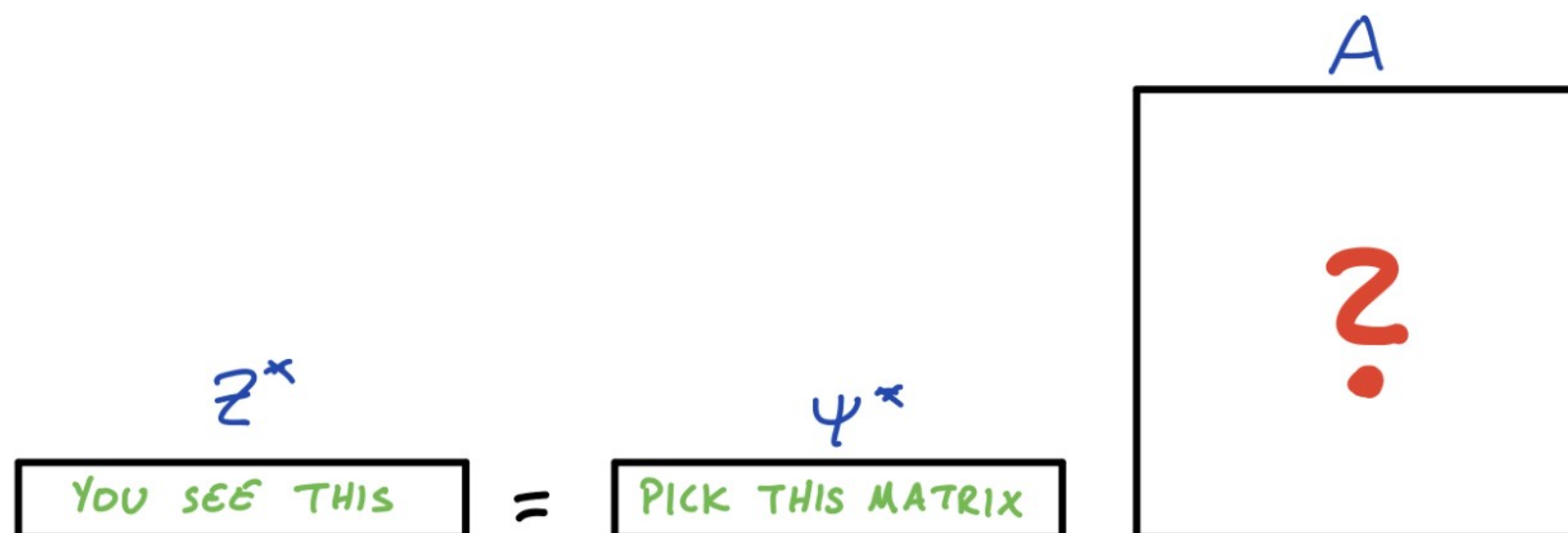
Environment: We are given a rank structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ (to be precise, \mathbf{A} is HBS/HSS of rank k). We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

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

Sample the column space of the matrix:



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



Compression of global operators – problem definition:

Environment: We are given a rank structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ (to be precise, \mathbf{A} is HBS/HSS of rank k). We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

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

Available techniques for the rank structured case:

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

- L. Lin, J. Lu, L. Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, JCP 2011.
- P.G. Martinsson, SISC, **38**(4), pp. A1959-A1986, 2016.

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

Recently proposed algorithms have reduced the pre-factors by constructing bespoke random matrices that are designed to be optimal for any given tessellation pattern. The key technical idea is to formulate admissibility criteria that form a graph, and then exploit powerful graph coloring algorithms. [J. Levitt & P.G. Martinsson, *arXiv:2205.03406*, 2022.]

Compression of global operators – problem definition:

Environment: We are given a rank structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ (to be precise, \mathbf{A} is HBS/HSS of rank k). We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

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

Available techniques for the rank structured case:

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

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

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

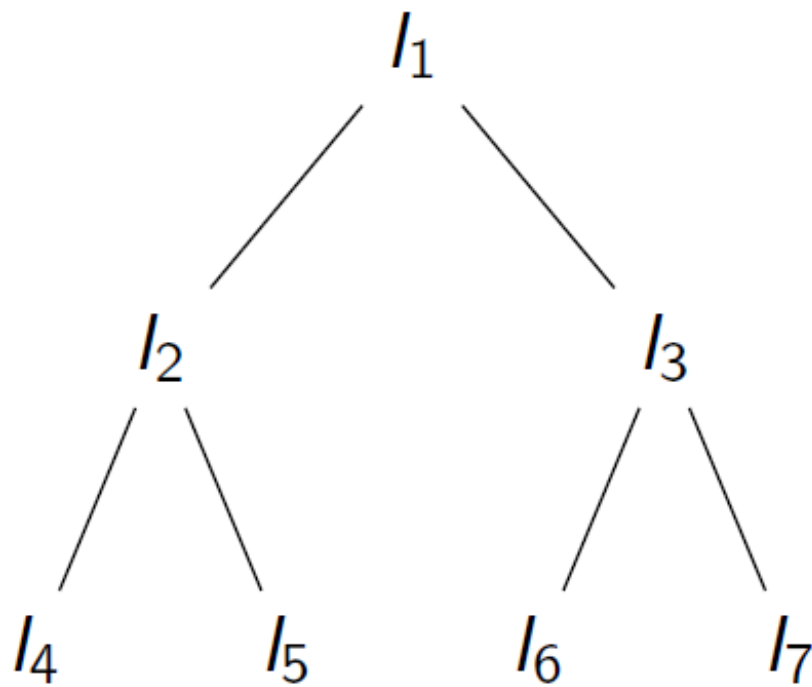
- P.G. Martinsson, SIMAX, **32**(4), 2011.
- Later improvements by Jianlin Xia, Sherry Li, and others. Widely used.

However, a fully black box algorithm with true linear complexity and high practical speed is now available:

- J. Levitt & P.G. Martinsson, arxiv arXiv:2205.02990, 2022.

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 $[1, \dots, 400]$.



Level 0: $l_1 = [1, 2, \dots, 400]$

Level 1:

$l_2 = [1, \dots, 200]$, $l_3 = [201, \dots, 400]$

Level 2:

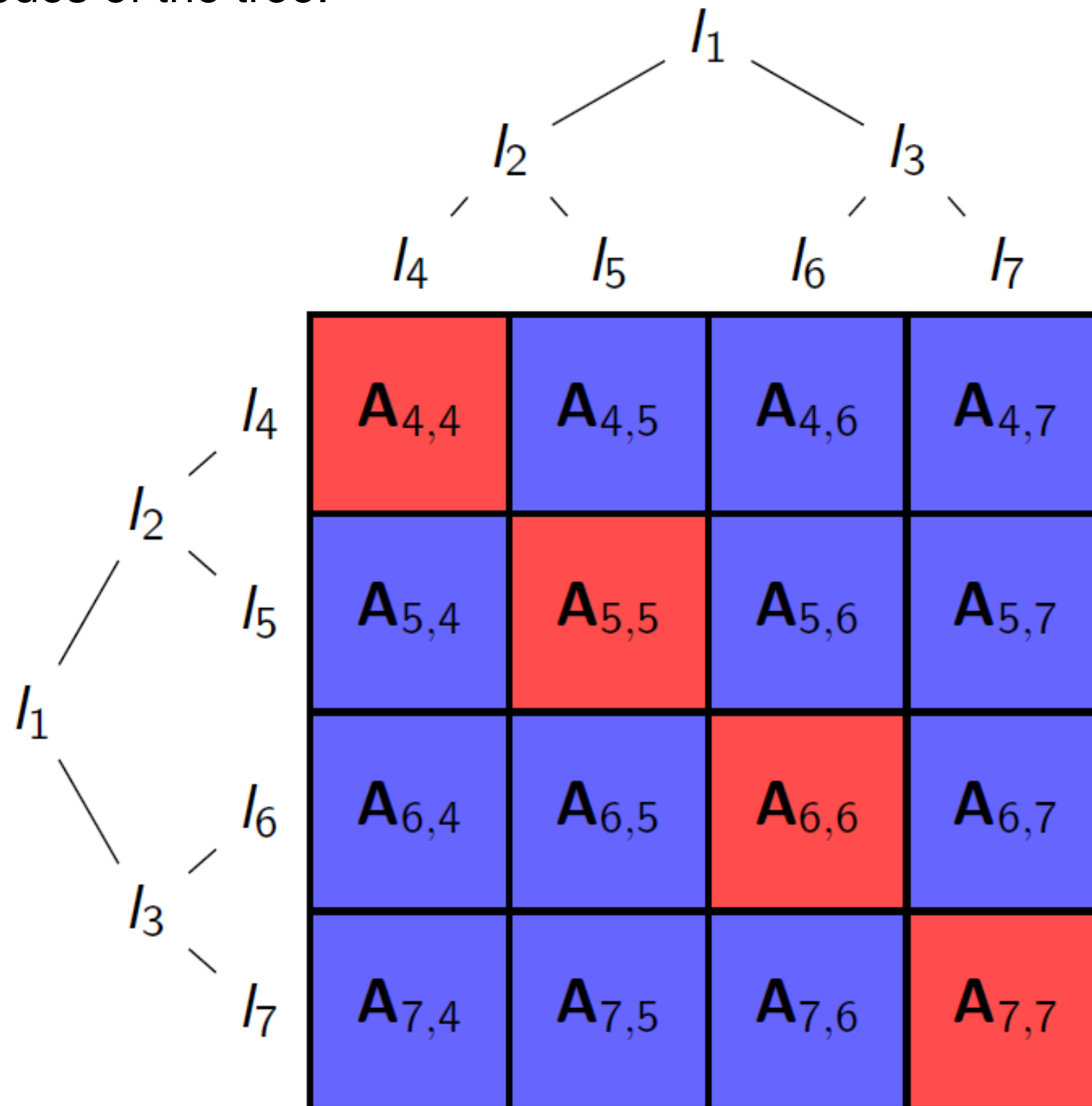
$l_4 = [1, \dots, 100]$, $l_5 = [101, \dots, 200]$,
 $l_6 = [201, \dots, 300]$, $l_7 = [301, \dots, 400]$

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}_τ and \mathbf{V}_τ such that for every leaf node $\tau' \neq \tau$, we have

$$\mathbf{A}_{\tau,\tau'} = \mathbf{U}_\tau \tilde{\mathbf{A}}_{\tau,\tau'} \mathbf{V}_{\tau'}^*.$$

$m \times m \quad m \times k \quad k \times k \quad k \times m$

$\mathbf{A}_{4,4}$	$\mathbf{A}_{4,5}$	$\mathbf{A}_{4,6}$	$\mathbf{A}_{4,7}$
$\mathbf{A}_{5,4}$	$\mathbf{A}_{5,5}$	$\mathbf{A}_{5,6}$	$\mathbf{A}_{5,7}$
$\mathbf{A}_{6,4}$	$\mathbf{A}_{6,5}$	$\mathbf{A}_{6,6}$	$\mathbf{A}_{6,7}$
$\mathbf{A}_{7,4}$	$\mathbf{A}_{7,5}$	$\mathbf{A}_{7,6}$	$\mathbf{A}_{7,7}$

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}_τ and \mathbf{V}_τ such that for every leaf node $\tau' \neq \tau$, we have

$$\mathbf{A}_{\tau,\tau'} = \mathbf{U}_\tau \tilde{\mathbf{A}}_{\tau,\tau'} \mathbf{V}_{\tau'}^*.$$

$m \times m \quad m \times k \quad k \times k \quad k \times m$

$\mathbf{A}_{4,4}$	$\mathbf{A}_{4,5}$	$\mathbf{A}_{4,6}$	$\mathbf{A}_{4,7}$
$\mathbf{A}_{5,4}$	$\mathbf{A}_{5,5}$	$\mathbf{A}_{5,6}$	$\mathbf{A}_{5,7}$
$\mathbf{A}_{6,4}$	$\mathbf{A}_{6,5}$	$\mathbf{A}_{6,6}$	$\mathbf{A}_{6,7}$
$\mathbf{A}_{7,4}$	$\mathbf{A}_{7,5}$	$\mathbf{A}_{7,6}$	$\mathbf{A}_{7,7}$

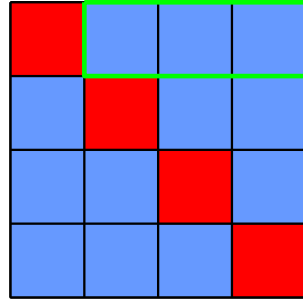
The on-diagonal blocks are not assumed to be low-rank, and pose the main challenge for black-box compression.

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

Approximation of rank-structured matrices – A naive approach

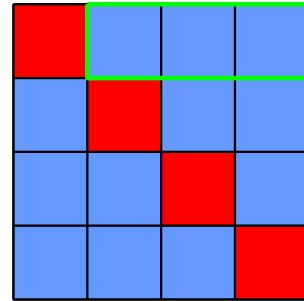
Consider the task of finding a basis matrix \mathbf{U}_4 for node 4 using randomized sampling.

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

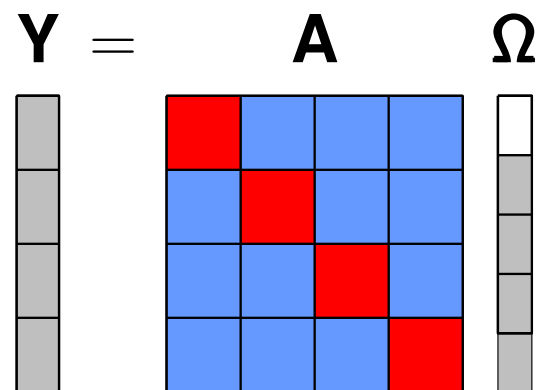


Approximation of rank-structured matrices – A naive approach

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.

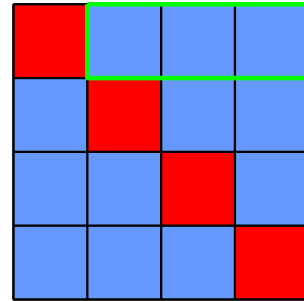


The naive approach is to sample with a random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4 . Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^C)$.

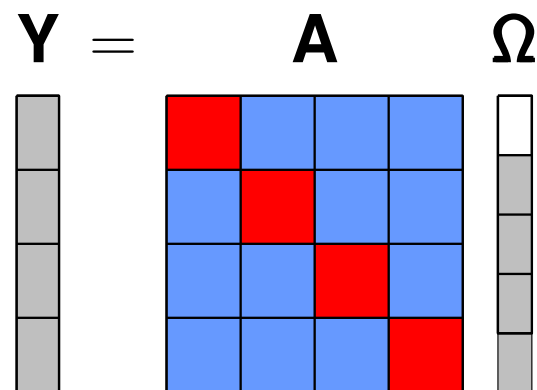


Approximation of rank-structured matrices – A naive approach

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.



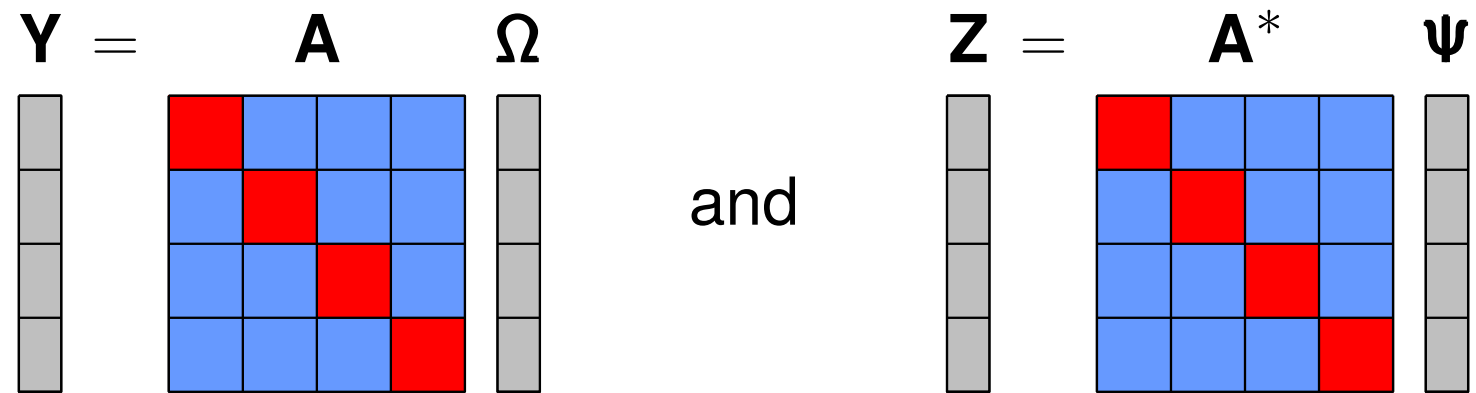
The naive approach is to sample with a random matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4 . Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^C)$.



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

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 $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:

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

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

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

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

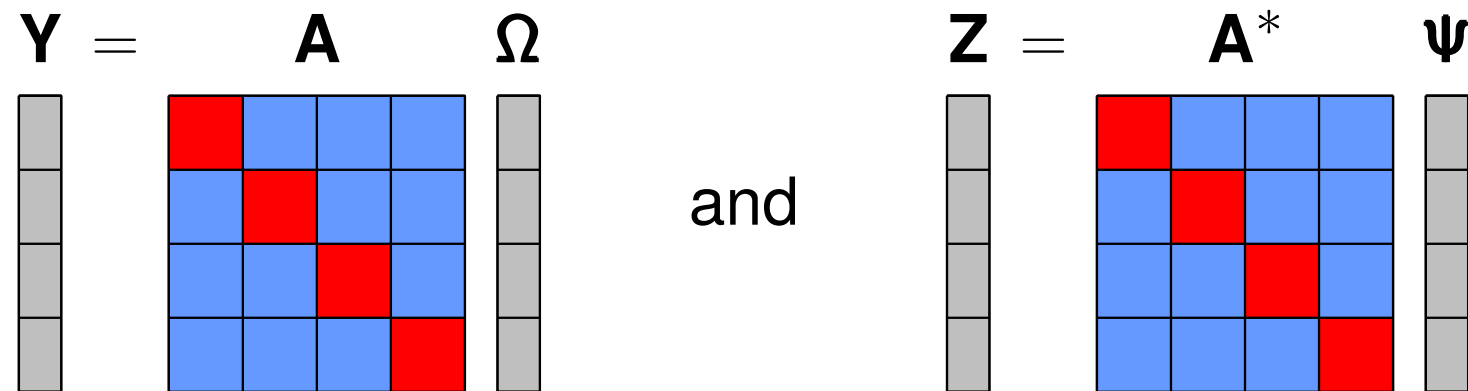
Processing \mathbf{Z} analogously, we obtain basis matrices \mathbf{Y}_j and \mathbf{Z}_j for $j \in \{4, 5, 6, 7\}$ such that

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

for *some* small matrices $\mathbf{B}_{i,j}$. How do you find them?

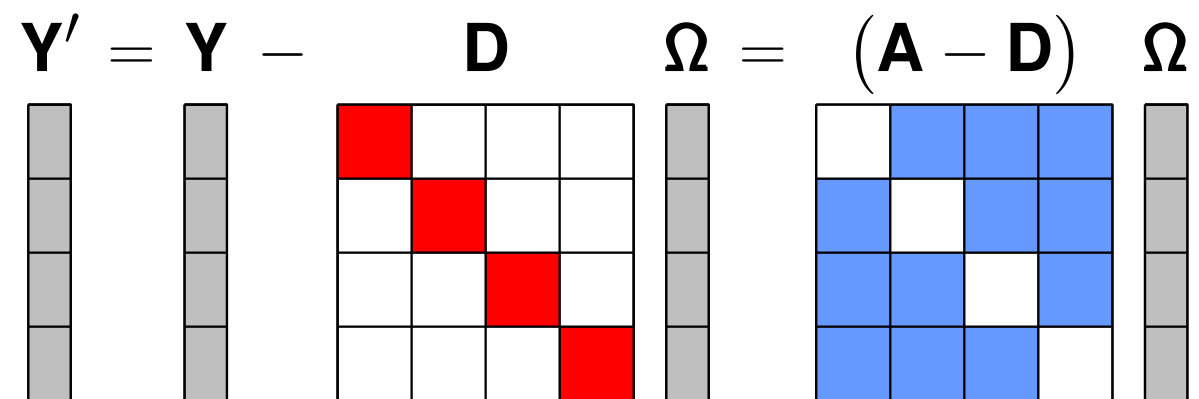
Approximation of rank-structured matrices – the “almost” black-box case

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.)

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



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

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

for *some* small matrices $\mathbf{B}_{i,j}$. How do you find them?

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 $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:

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

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

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

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

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

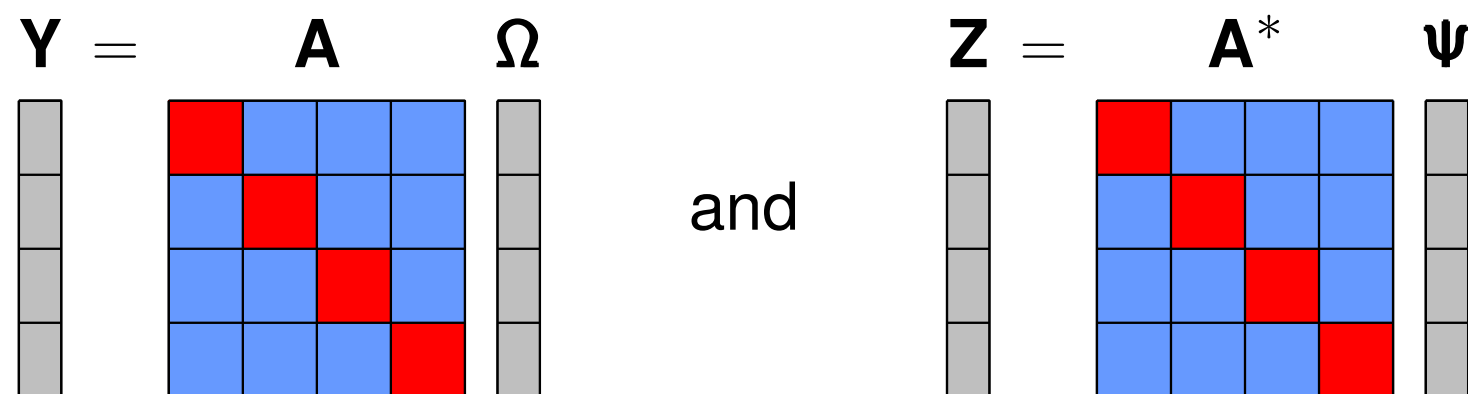
$$\mathbf{A}_{i,j} \approx \mathbf{Y}_i \mathbf{B}_{i,j} \mathbf{Z}_j^*, \quad i \neq j,$$

for *some* small matrices $\mathbf{B}_{i,j}$. How do you find them? Perform IDs on \mathbf{Y}_i and \mathbf{Z}_j :

$$\mathbf{A}_{i,j} \approx \hat{\mathbf{Y}}_i \mathbf{A}(I_i^S, J_j^S) \hat{\mathbf{Z}}_j^*, \quad i \neq j,$$

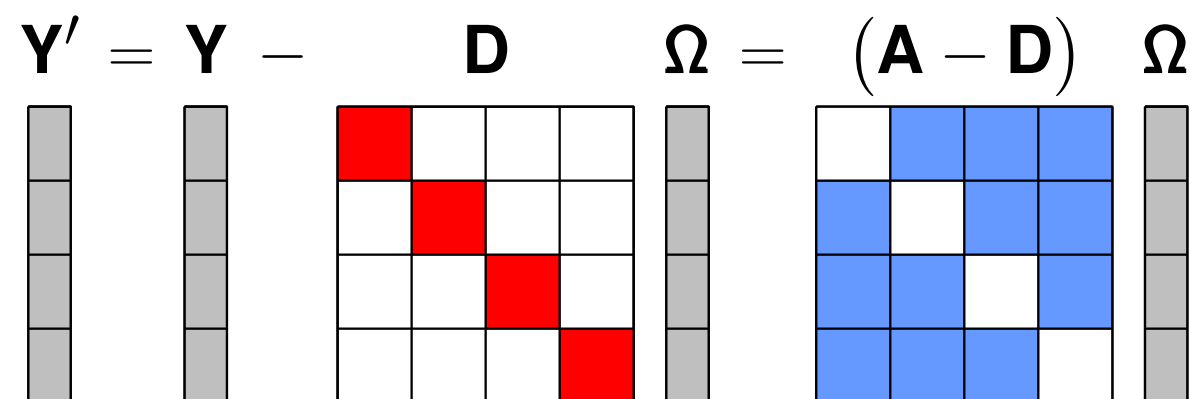
Approximation of rank-structured matrices – the “almost” black-box case

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.)

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



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

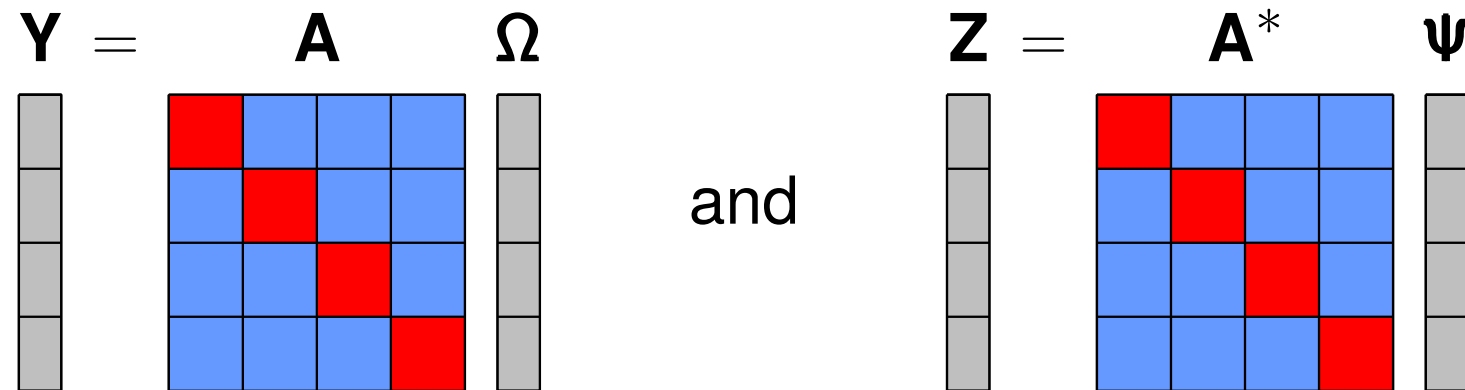
$$\mathbf{A}_{i,j} \approx \mathbf{Y}_i \mathbf{B}_{i,j} \mathbf{Z}_j^*, \quad i \neq j,$$

for *some* small matrices $\mathbf{B}_{i,j}$. How do you find them? Perform IDs on \mathbf{Y}_i and \mathbf{Z}_j :

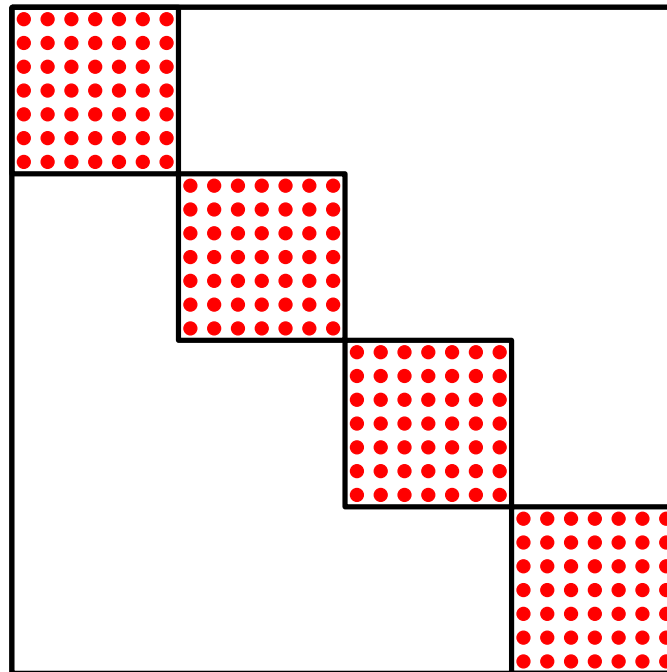
$$\mathbf{A}_{i,j} \approx \hat{\mathbf{Y}}_i \mathbf{A}(I_i^S, J_j^S) \hat{\mathbf{Z}}_j^*, \quad i \neq j, \quad \text{Only need to evaluate } \mathbf{A}(I_i^S, J_j^S)!$$

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

Step 1: Sample \mathbf{A} with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:

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


Step 2: Construct the diagonal blocks of the matrix explicitly.

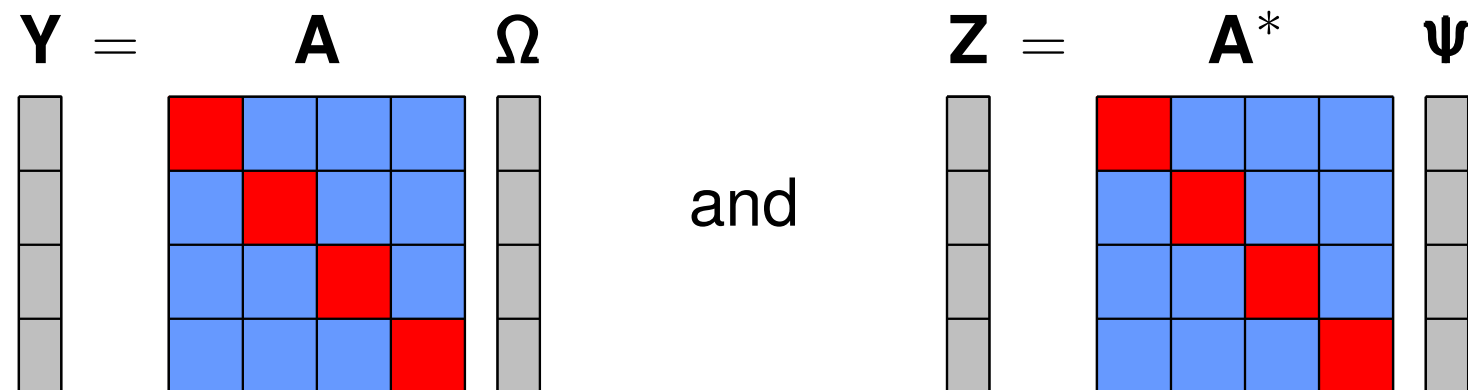


Having formed the diagonal blocks, we eliminate their contributions to \mathbf{Y} and \mathbf{Z} to get “pure” samples from off-diagonal blocks.

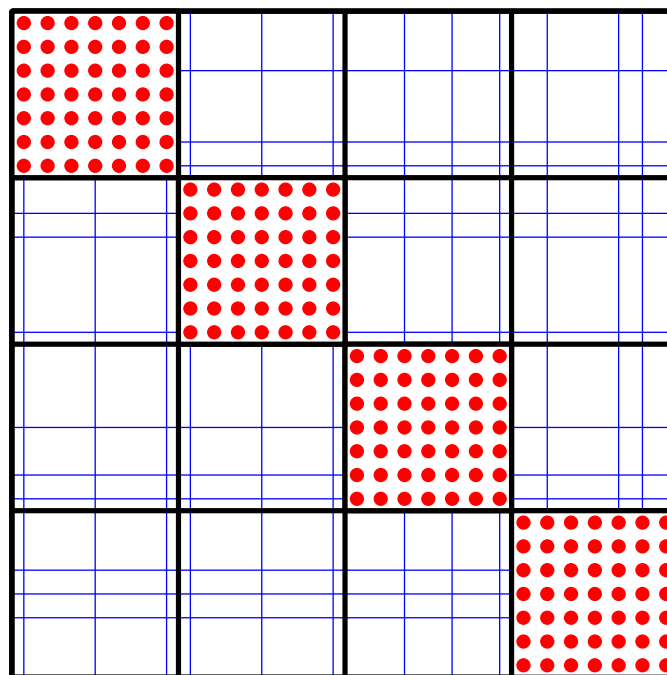
Then determine the spanning rows and columns in each off-diagonal block.

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

Step 1: Sample \mathbf{A} with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:

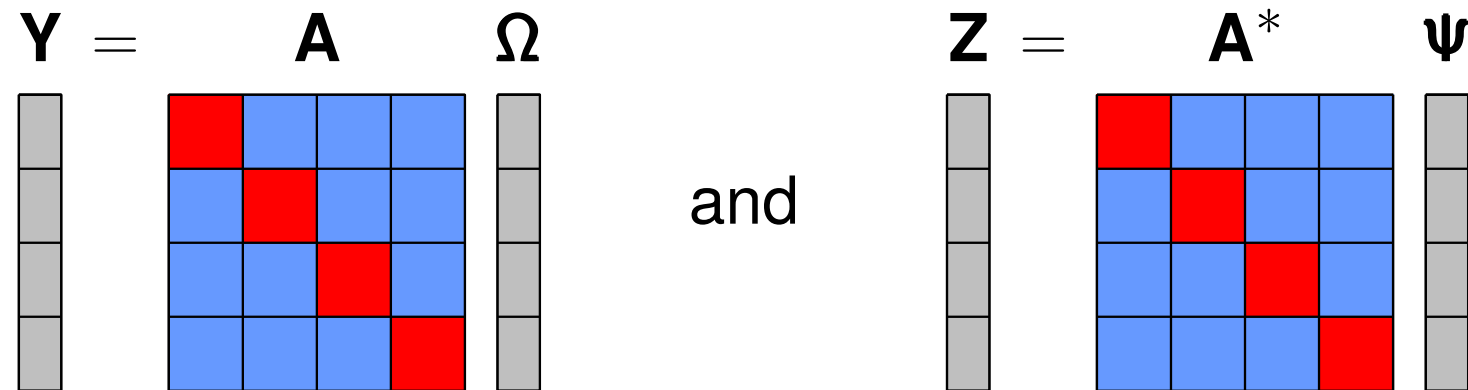


Step 3: We now know the spanning columns and the spanning rows.

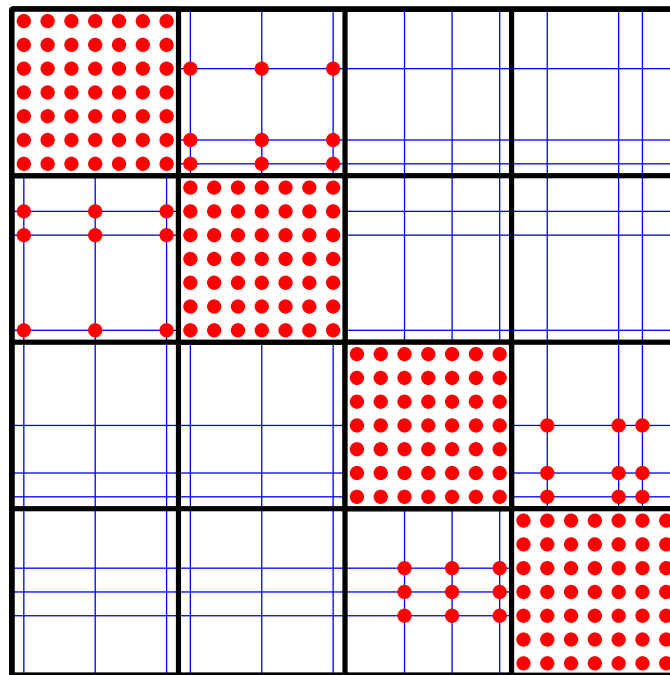


Approximation of rank-structured matrices – the “almost” black-box case

Step 1: Sample \mathbf{A} with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:



Step 3: We now know the spanning columns and the spanning rows.

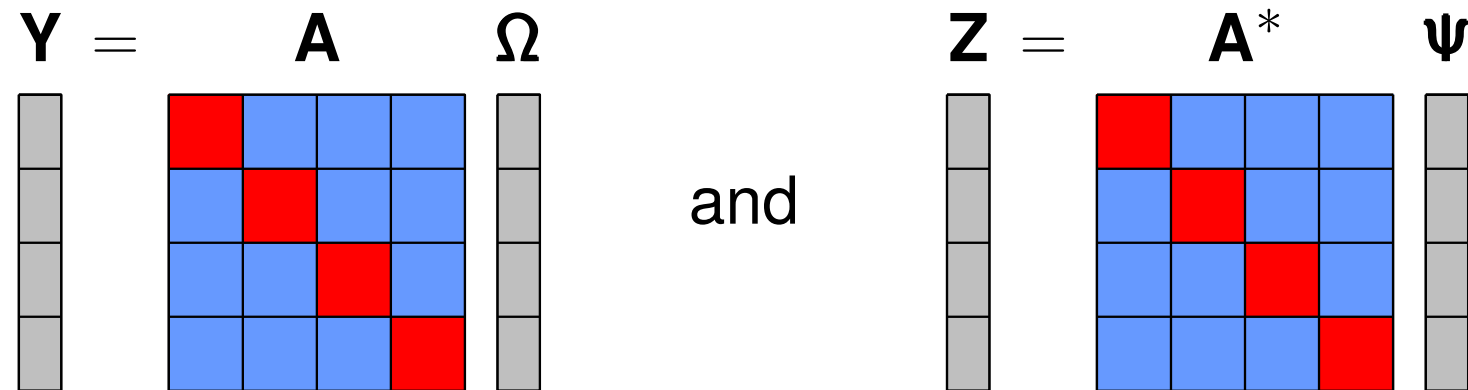


Evaluate the entries of \mathbf{A} that lie at the intersections.

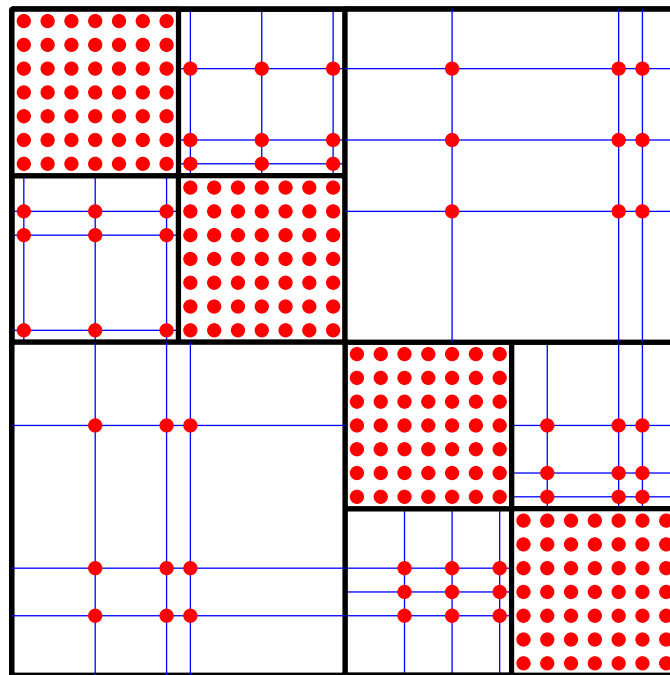
Do this *only for the blocks that are needed to form the diagonal blocks at the next level!*

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

Step 1: Sample \mathbf{A} with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N \times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N \times r}$:



Step 4: For the higher levels, recursively apply the same procedure!!



The number of entry evaluations gets cut in half at each coarser level

→ $O(Nk)$ entry evaluations required.

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

Computational costs of scheme described:

- Applications of \mathbf{A} and \mathbf{A}^* to Gaussian matrices with $\approx k + 10$ columns each.
- $O(Nk)$ entry evaluations.
- $O(Nk)$ storage.
- $O(Nk^2)$ flops.

Very fast in practice.

But: Requires the ability to evaluate entries!

Let us next lift that restriction.

Approximation of rank-structured matrices – fully black box

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

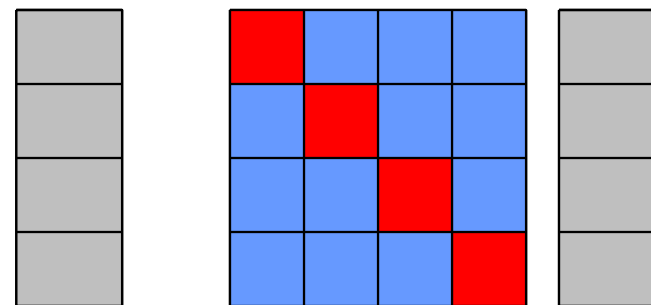
$$\mathbf{Y} = \mathbf{A} \Omega$$

The diagram shows three matrices: \mathbf{Y} , \mathbf{A} , and Ω . \mathbf{Y} is a vertical column of four gray boxes. \mathbf{A} is a 4x4 grid where the main diagonal elements are red and the off-diagonal elements are blue. Ω is a vertical column of four gray boxes.

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

Approximation of rank-structured matrices – fully black box

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

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


The diagram shows three matrices: \mathbf{Y} , \mathbf{A} , and Ω . \mathbf{Y} is a 4x4 grid of gray cells. \mathbf{A} is a 4x4 grid with a diagonal of red cells and blue cells elsewhere. Ω is a 4x4 grid of gray cells.

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

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

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

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

Approximation of rank-structured matrices – fully black box

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

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

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

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

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

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

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

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

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

Approximation of rank-structured matrices – fully black box

- For each leaf node τ , we compute

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

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

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

Approximation of rank-structured matrices – fully black box

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

Steps:

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

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

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

Compress level by level from finest to coarsest.

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

Approximation of rank-structured matrices – Finding \mathbf{D}

From the telescoping factorization

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

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

Approximation of rank-structured matrices – Finding \mathbf{D}

From the telescoping factorization

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

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

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

Approximation of rank-structured matrices – Finding \mathbf{D}

From the telescoping factorization

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

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

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

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

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

Approximation of rank-structured matrices – Compressing $\tilde{\mathbf{A}}^{(L)}$

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

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

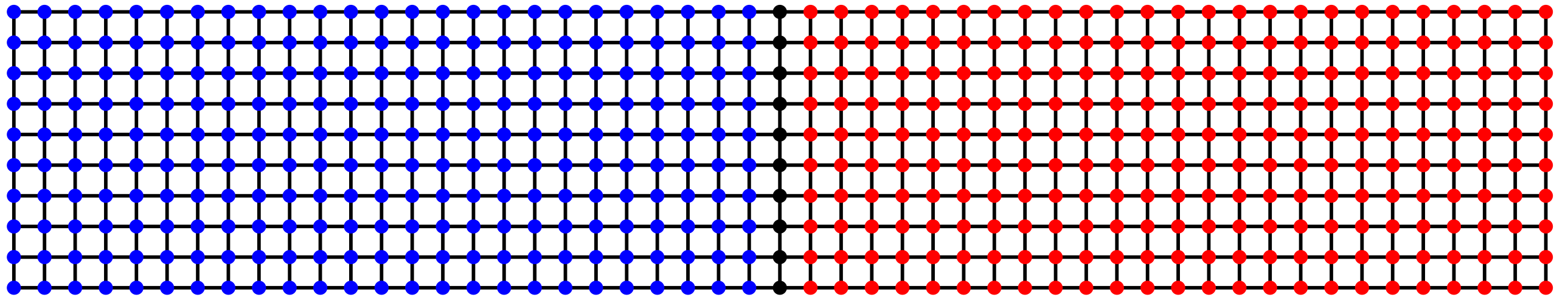
and rearrange to obtain

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

Approximation of rank-structured matrices: Sparse LU

Let \mathbf{C} be the stiffness matrix for the standard five-point stencil finite difference approximation to the Poisson equation on a rectangular grid.

We partition the grid as shown and tessellate \mathbf{C} accordingly.



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

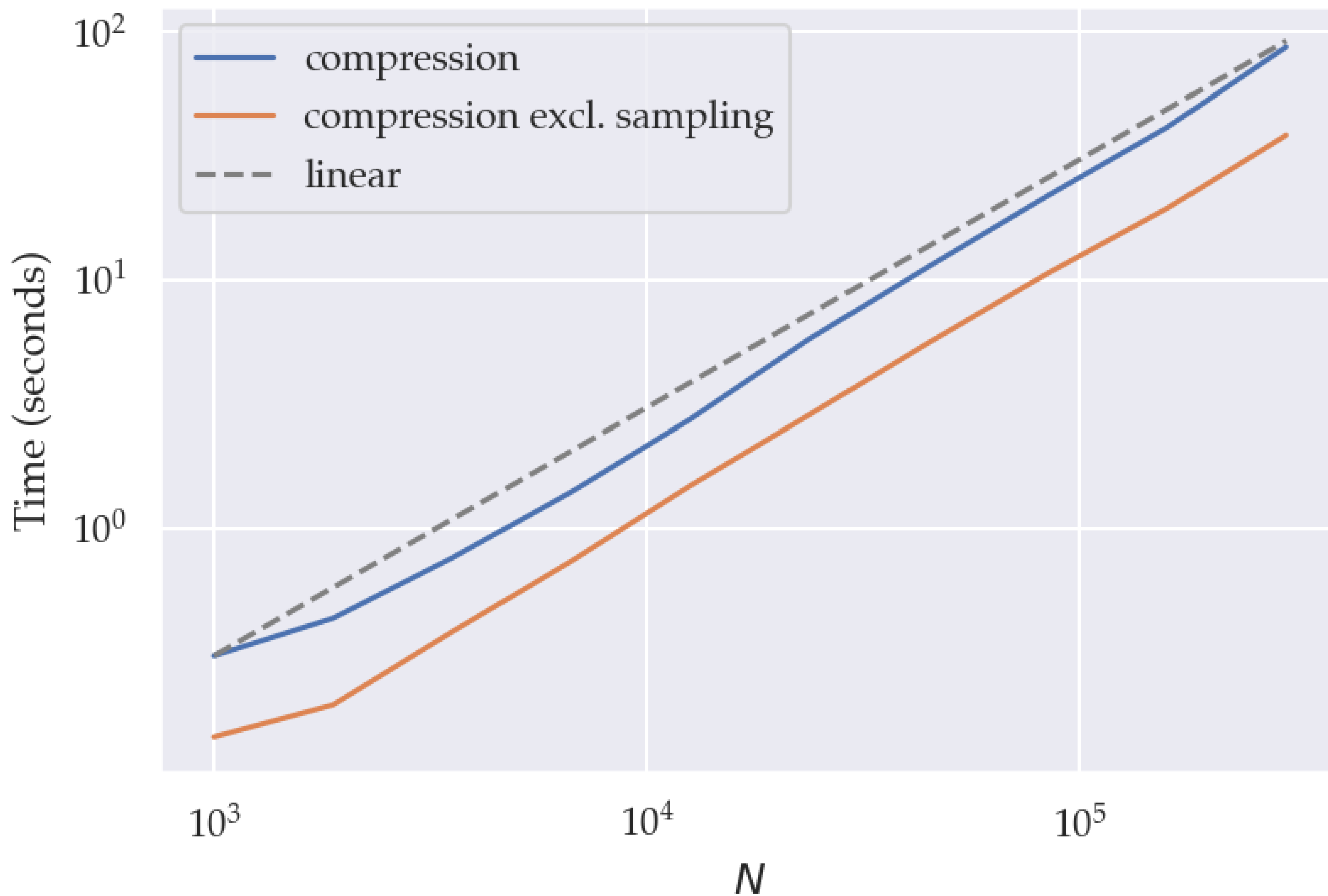
The matrix we seek to compress is the Schur complement

$$\mathbf{A} = \mathbf{C}_{33} - \mathbf{C}_{31}\mathbf{C}_{11}^{-1}\mathbf{C}_{31} - \mathbf{C}_{32}\mathbf{C}_{22}^{-1}\mathbf{C}_{23}.$$

Approximation of rank-structured matrices: Sparse LU

$r = 30, m = 60$

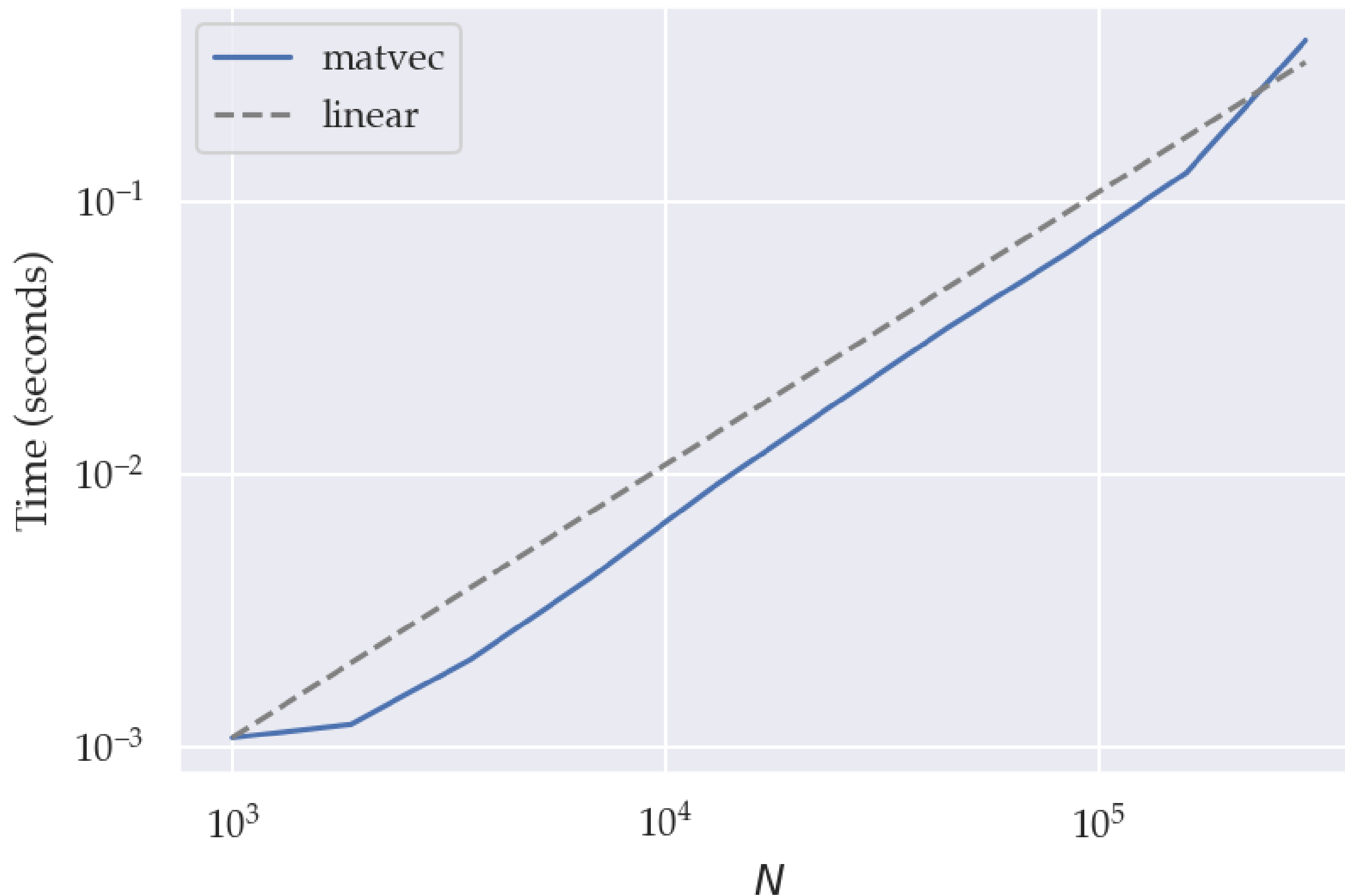
Time for matrix compression



Approximation of rank-structured matrices: Sparse LU

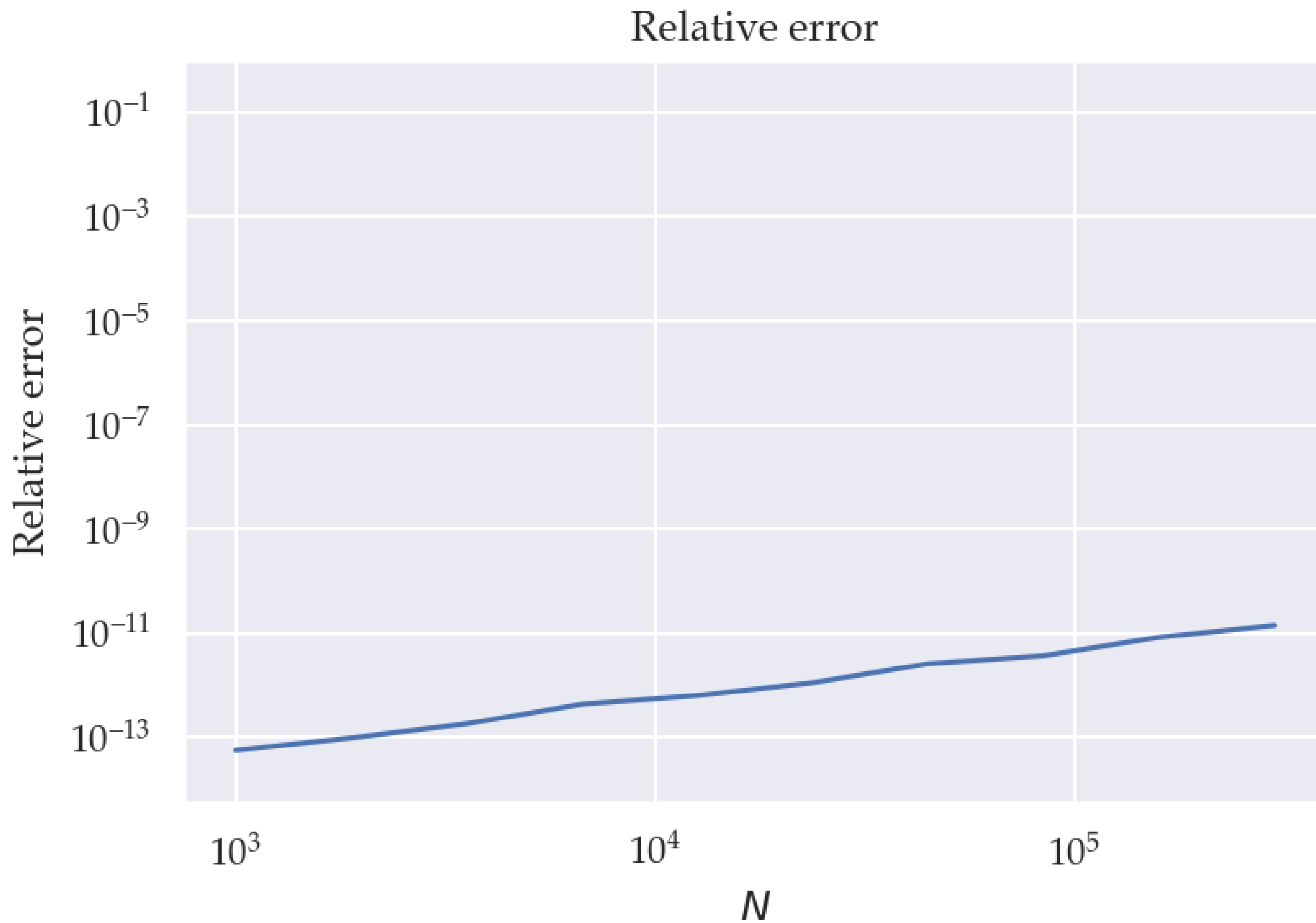
$r = 30, m = 60$

Time for matrix-vector multiplication



Approximation of rank-structured matrices: Sparse LU

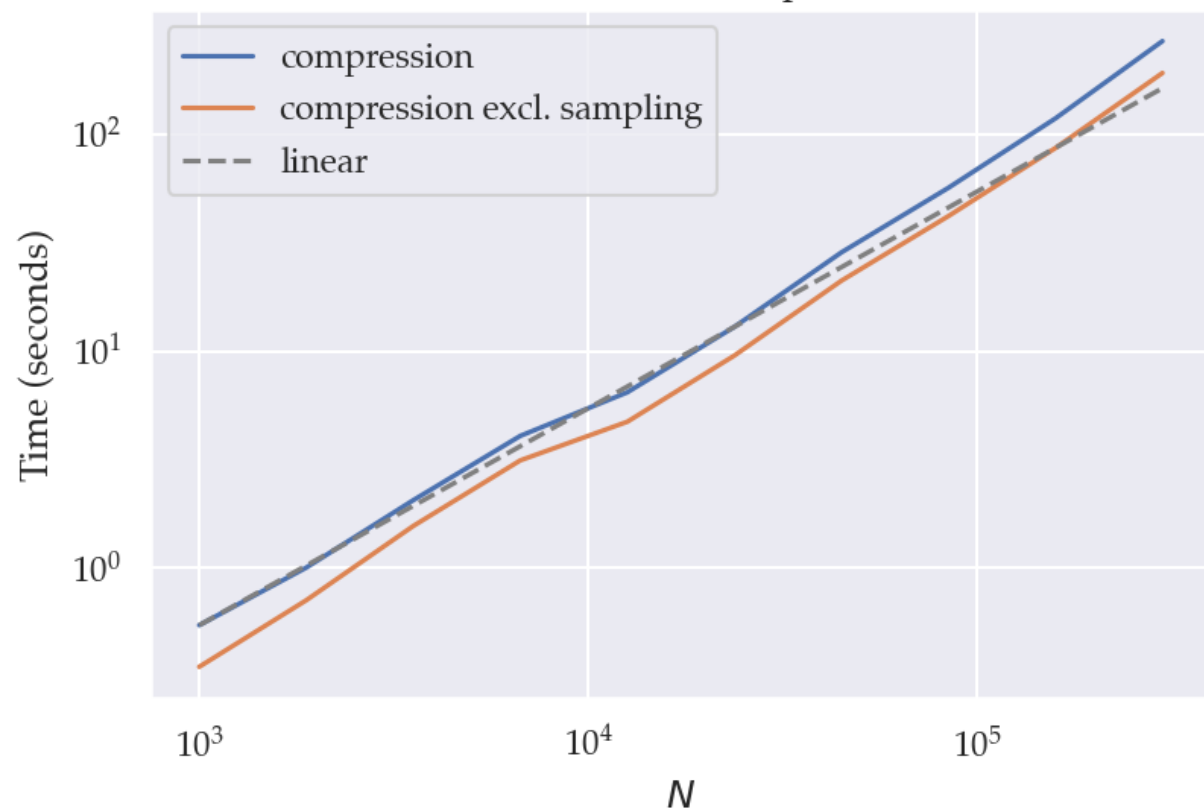
$r = 30, m = 60$



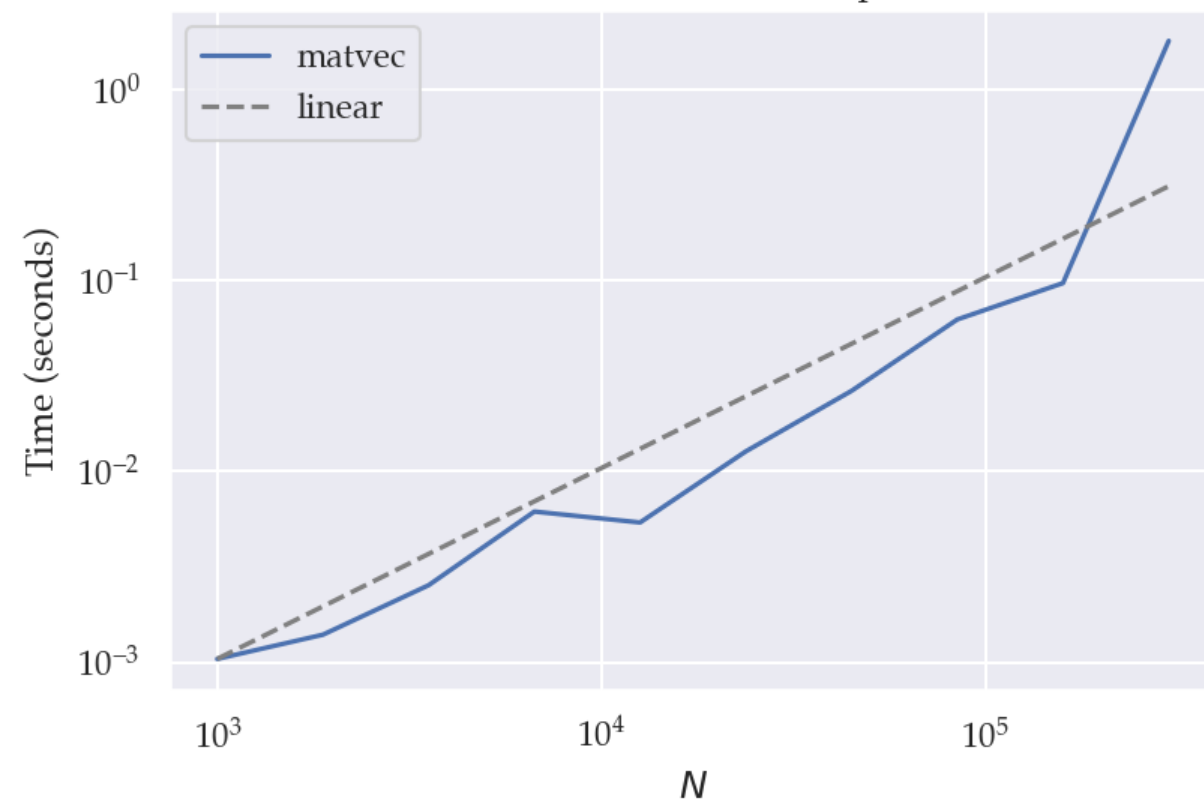
Approximation of rank-structured matrices: FMM

$r = 50, m = 100$

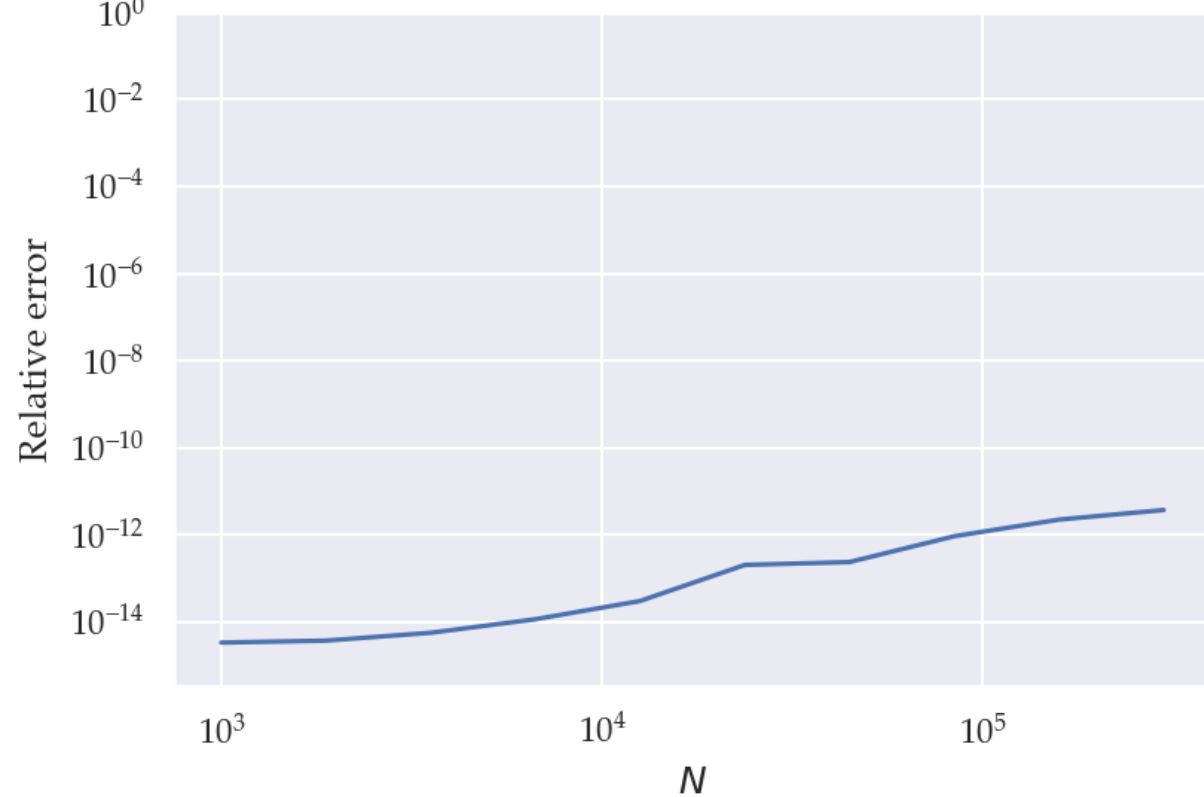
Time for matrix compression



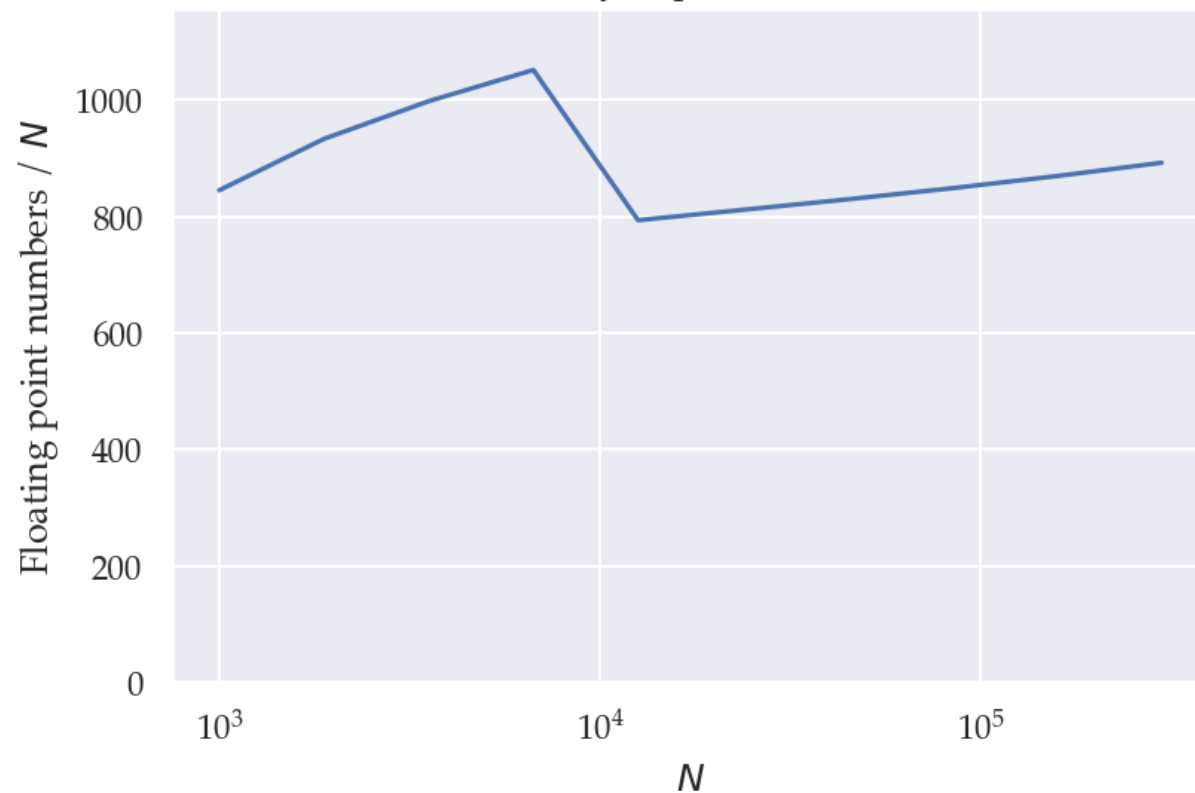
Time for matrix-vector multiplication



Relative error



Memory requirement / N



Approximation of rank-structured matrices: Key points

- Fully “black box”. Interacts with \mathbf{A} only via the matvec.
- True linear complexity. Requires only $O(k)$ samples from \mathbf{A} and \mathbf{A}^* .
Much faster in practice than existing black box algorithms.
- Ideal tool for acceleration of sparse direct solvers.
- All matrix-vector multiplies can be done in one go. *Streaming algorithm!*

Key points:

- Interpolatory and CUR decompositions are useful and popular.
 - Preserve properties like sparsity and non-negativity. Good for data interpretation.
 - Storage efficient.
 - Invaluable in the context of modern Fast Multipole Methods and Fast Direct Solvers.
 - Enables divide-and-conquer & localization. ***Essential for 3D computations!*** (Betcke, Wathen)
- Randomized sketching is an excellent tool for computing the CUR/ID.
 - Particularly effective for large sparse matrices, and huge matrices stored out of core.
 - Improved asymptotic flop count — “fast Johnson-Lindenstrauss transforms”.
 - As robust and accurate as deterministic methods.

Keep it simple, however! Gaussian sketch + partially pivoted LU excel together.

- Randomized algorithms for compressing rank structured matrices.
 - Unlocks \mathcal{H} -matrix arithmetic in any situation where you have access to fast application of the operator to vectors.
 - Key application: Acceleration of sparse direct solvers to close to linear complexity.
- The ideas presented were strongly influenced by Nick’s work. Thank you!
 - Application of powerful ideas from (dense) linear algebra to solving continuum problems. Backwards stability. The value of high accuracy methods (error estimates less necessary when you have 15 correct digits). An ideal to strive for in writing (very hard to mimic, however!).

Slides: http://users.oden.utexas.edu/~pgm/main_talks/

Surveys:

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

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

- P.G. Martinsson, “Randomized methods for matrix computations.” *The Mathematics of Data*, IAS/Park City Mathematics Series, 25(4), pp. 187 - 231, 2018.

Book chapter that is written to be accessible to a broad audience. Focused on practical aspects rather than theory.

- N. Halko, P.G. Martinsson, J. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.” *SIAM Review*, 53(2), 2011, pp. 217-288.

Survey that describes the randomized SVD and its variations.

Tutorials, summer schools, etc:

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

Software:

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