Structured matrix computations

Gunnar Martinsson

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

University of Texas at Austin

Students, postdocs, collaborators: Daniel Appelö, Chao Chen, Ke Chen, Simon Dirckx, Yijun Dong, Adrianna Gillman, Abinand Gopal, Joe Kileel, Joseph Kump, James Levitt, Yuji Nakatsukasa, Michael O'Neil, Joel Tropp, Kate Pearce, Heather Wilber, Bowei Wu, Anna Yesypenko.

Research support by:

Global operators in scientific computing

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

Examples:

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

Classical methods:

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

Global operators in scientific computing

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

Examples:

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

Possible connections to Machine Learning:

- Exploit the knowledge we have about multiresolution representations of linear operators to find effective ways to represent global operators in ML models.
- Extend existing capabilities to non-linear problems.
- Provide compressed representations (“reduced models”) for linear sub systems in multiscale or multiphysics simulations.

Global operators in scientific computing

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

Examples:

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

Objectives of this talk:

- Describe key properties of global operators of mathematical physics.
- Review key ideas from the classical literature (FMM, $\mathcal{H}$-matrices, etc).
- Describe recent work on randomized algorithms for efficiently computing compressed representations of global operators. “Operator learning”?

Starting point for discussions during the workshop. (No ML in this talk. . .)

Outline of talk

(1) **The role of global operators in scientific computing.**

(2) **Interaction ranks – why are they small?**

(3) **Introduction to rank structured matrices.**

(4) **Randomized method for compressing global operators.**

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

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

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

Examples of problems:

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

Archetypical example: Poisson equation with Dirichlet boundary data:

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

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

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

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

Examples of problems:

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

Archetypical example: Poisson equation with Dirichlet boundary data:

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

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

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

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

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

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

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

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

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

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

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

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

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

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

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

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

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

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

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

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

Bounded, smoothing, often fairly stable, etc.

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

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

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

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

Example: Solution operator to an elliptic PDE

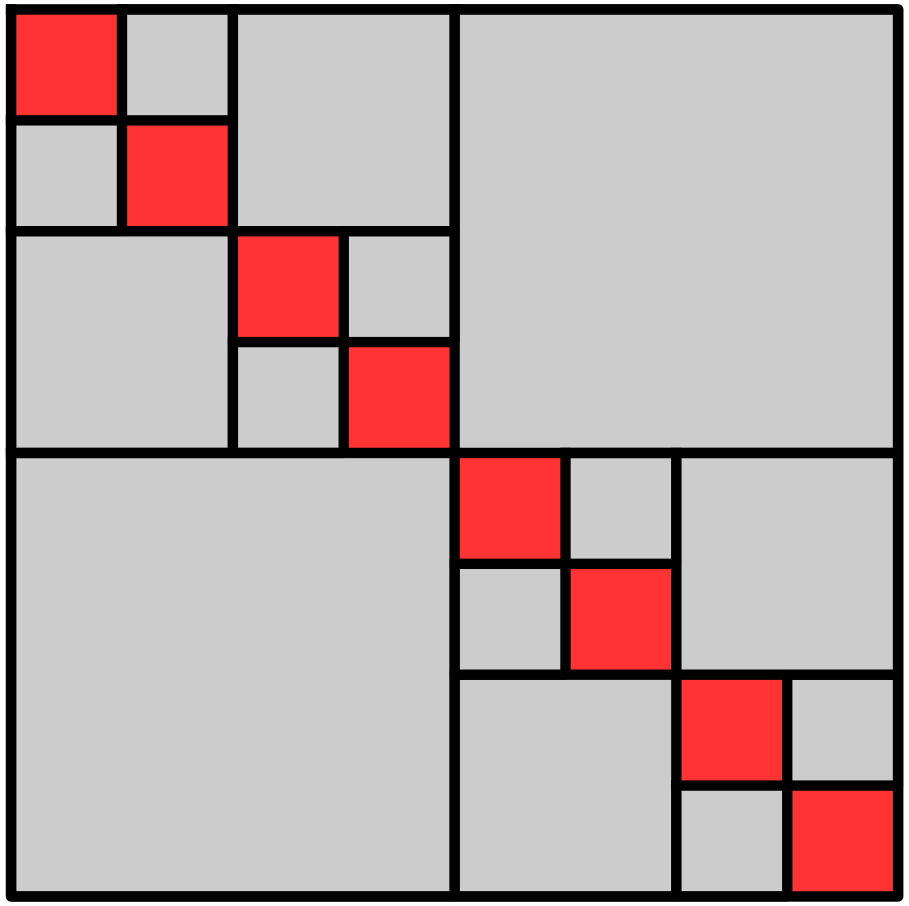
Consider a linear boundary value problem of the form

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

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

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

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



All gray blocks have low rank.

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

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

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

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

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

Example: Solution operator to an elliptic PDE

Consider a linear boundary value problem of the form

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

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

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

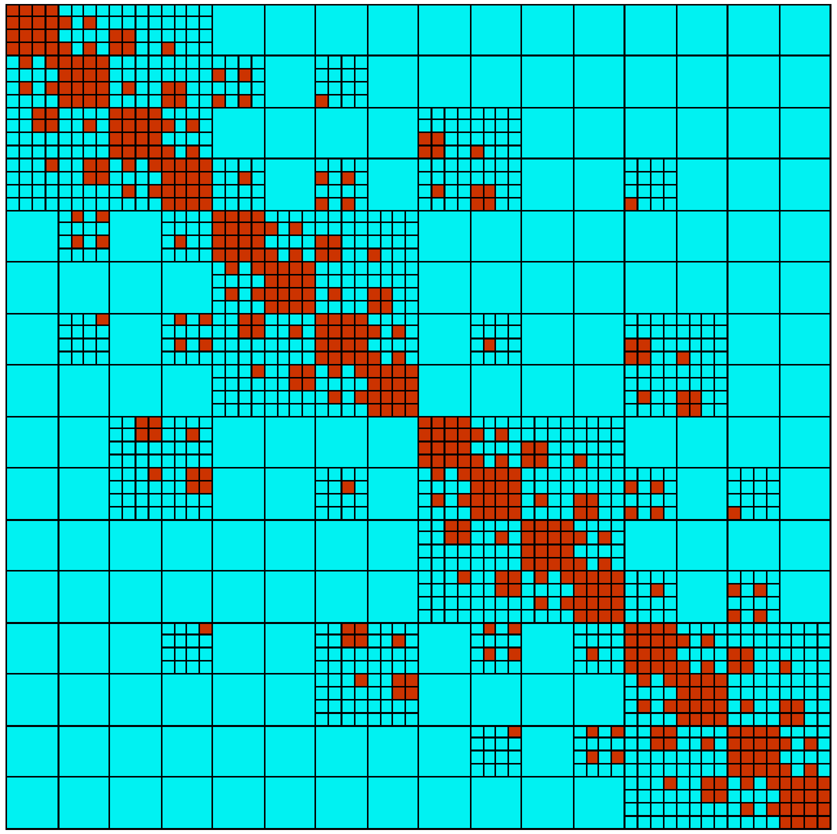
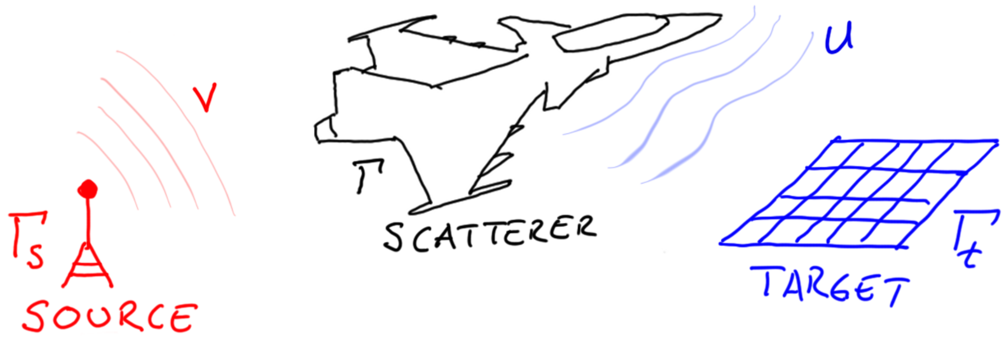


Image credit: Ambikasaran & Darve, arxiv.org #1407.1572

Example: Boundary integral equation

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



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

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

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

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

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

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

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

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

Consider a well posed linear boundary value problem

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

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

$$T : f \mapsto n$$

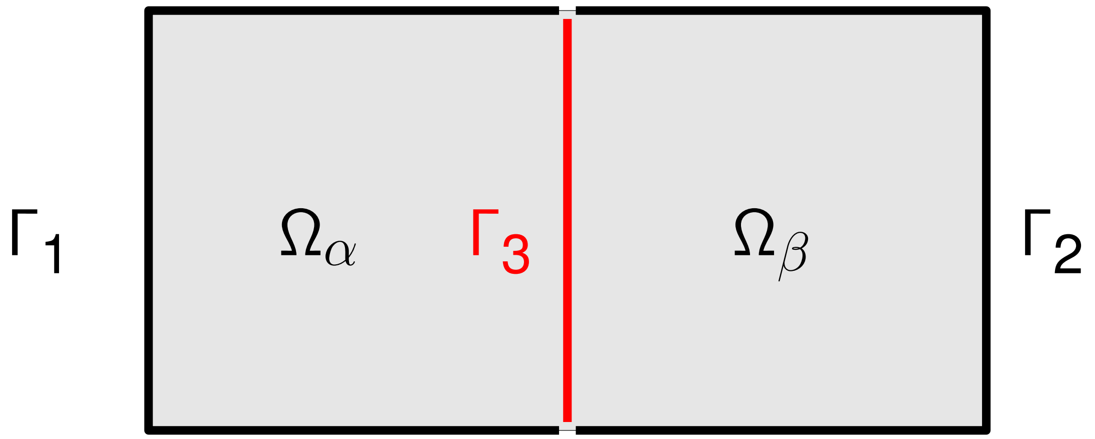
is known as the *Dirichlet-to-Neumann (DtN)* map.

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

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

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

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



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

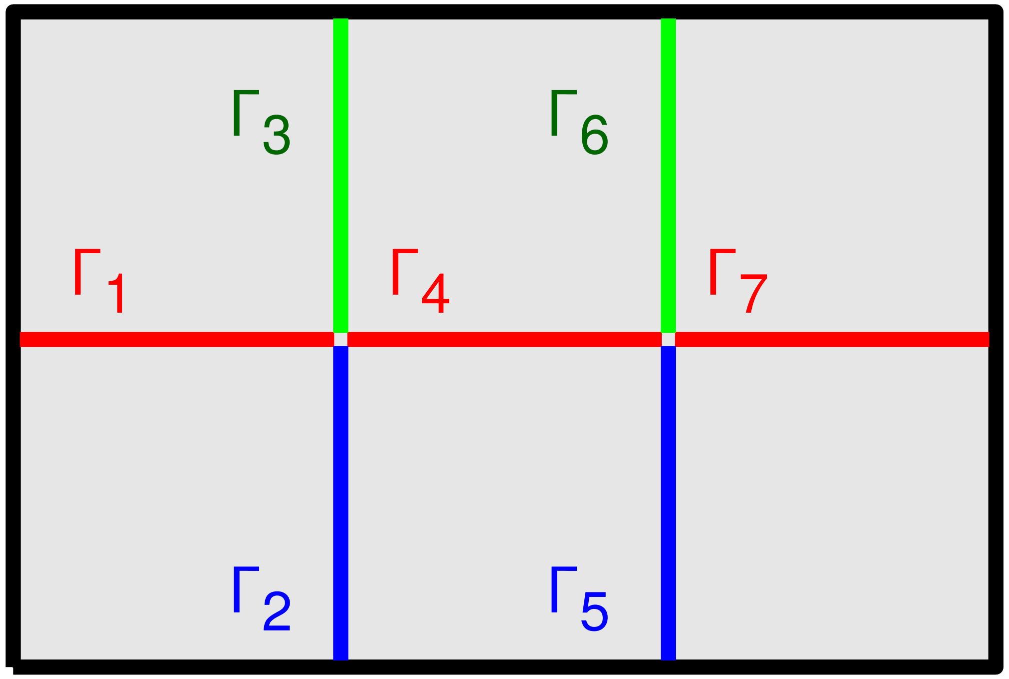
$$\begin{bmatrix} T^\alpha_{11} & T^\alpha_{13} \\ T^\alpha_{31} & T^\alpha_{33} \end{bmatrix} \begin{bmatrix} f_1 \\ f_3 \end{bmatrix} = \begin{bmatrix} n_1 \\ n^\alpha_3 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} T^\beta_{22} & T^\beta_{23} \\ T^\beta_{32} & T^\beta_{33} \end{bmatrix} \begin{bmatrix}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

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

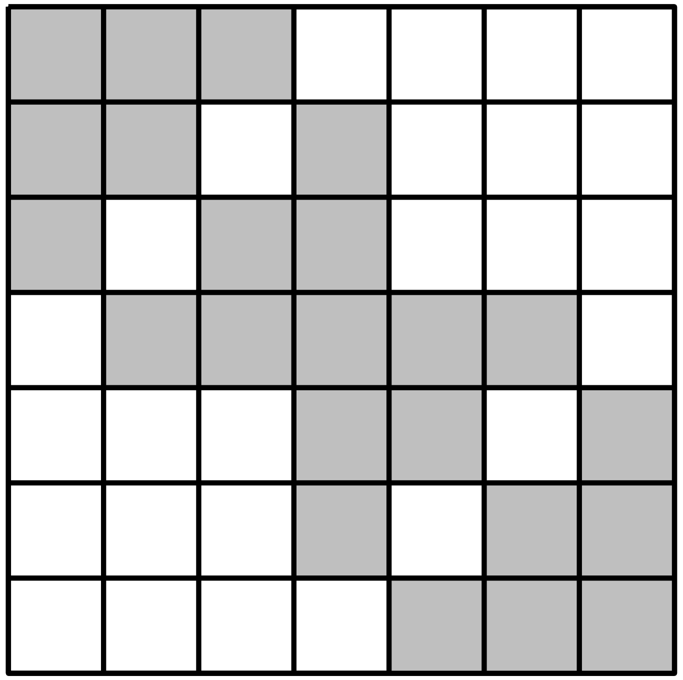
$$(T^\alpha_{33} + T^\beta_{33})f_3 = -T^\alpha_{31}f_1 - T^\beta_{32}f_2.$$

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

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



The domain.



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

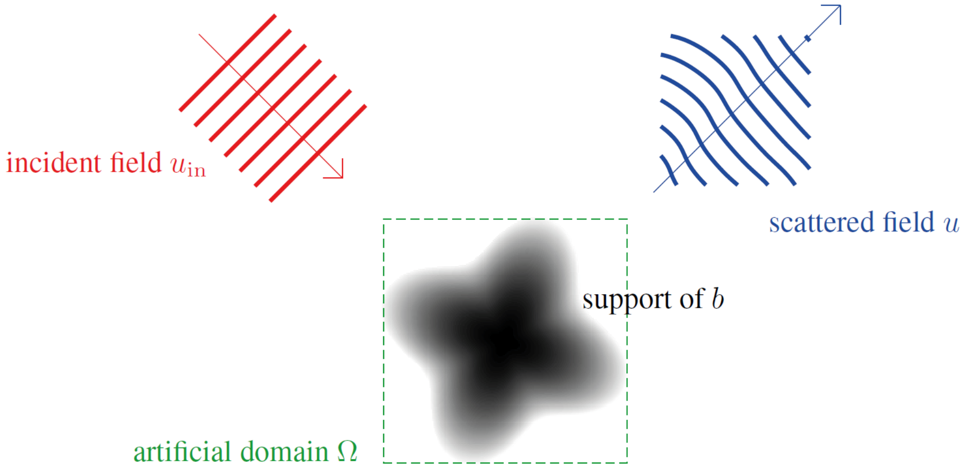
Example: Poincaré-Steklov operators (Dirichlet-to-Neumann, etc)

Consider the free space acoustic scattering problem

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

where

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



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

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

Glue the domains together using the DtNs.

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

[Gillman, Barnett, Martinsson, 2015]

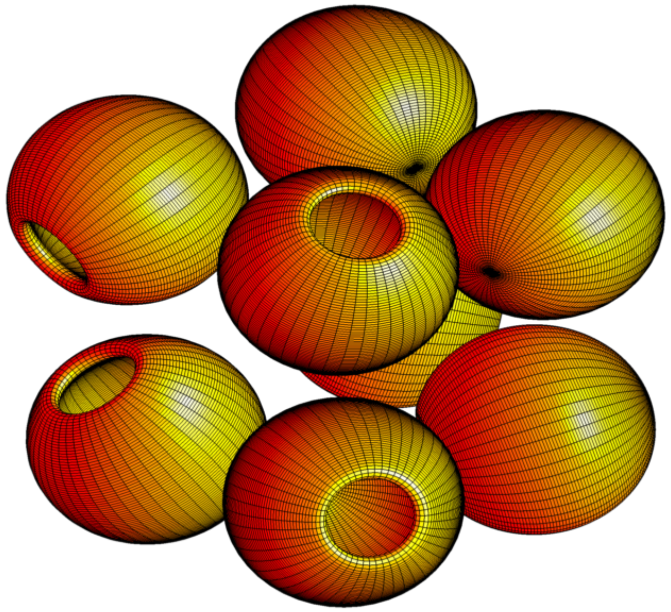
Example: Scattering operators

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

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

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

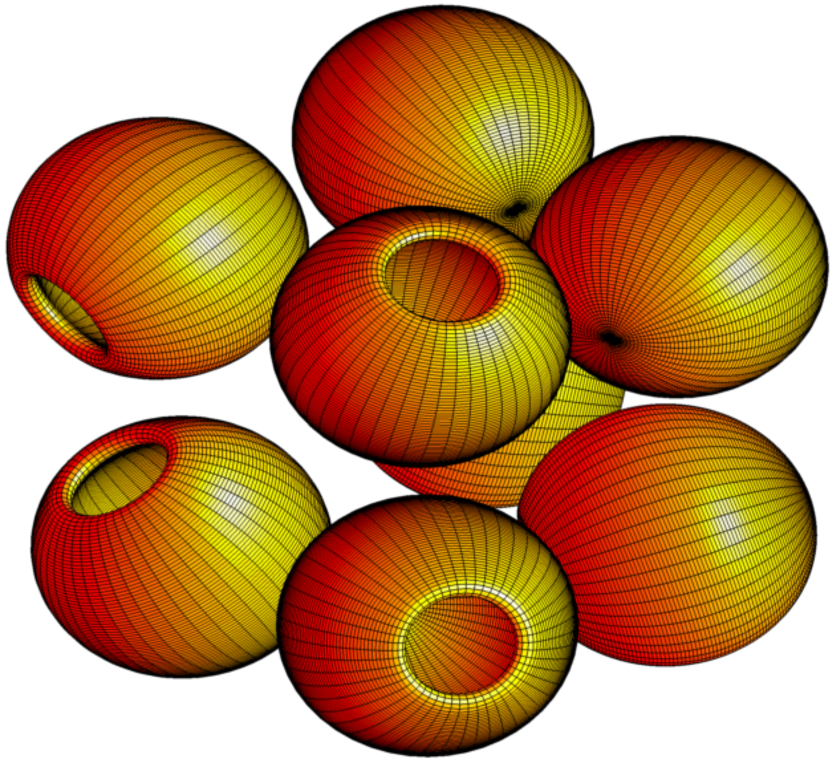
The benefit is that the global system you form this way is *far smaller* than a global system that fully resolves all individual scatters. (And often better conditioned too!)



Example: Scattering operators

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

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



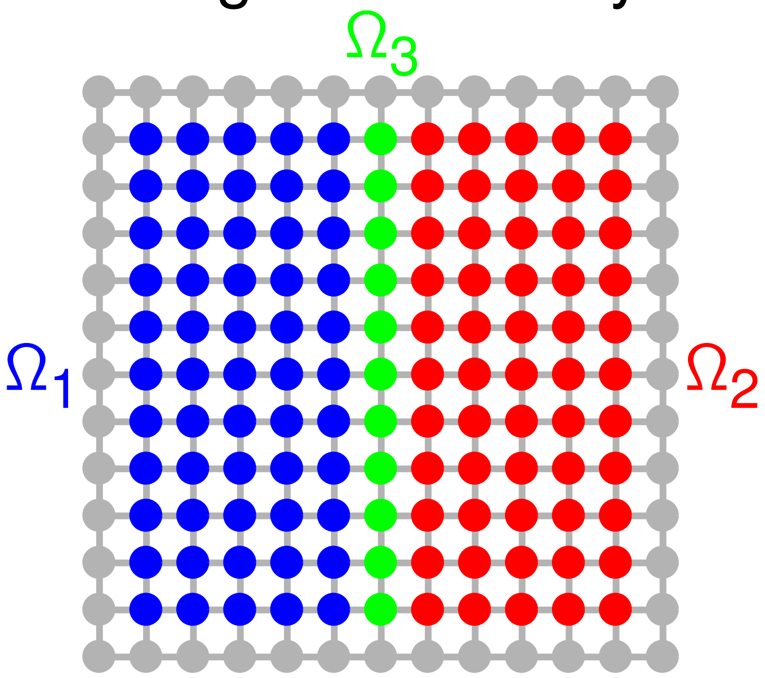
A hybrid direct/iterative solver is used (a highly accurate scattering matrix $\mathbf{S}_i$ is computed for body i) to form a global system

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

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

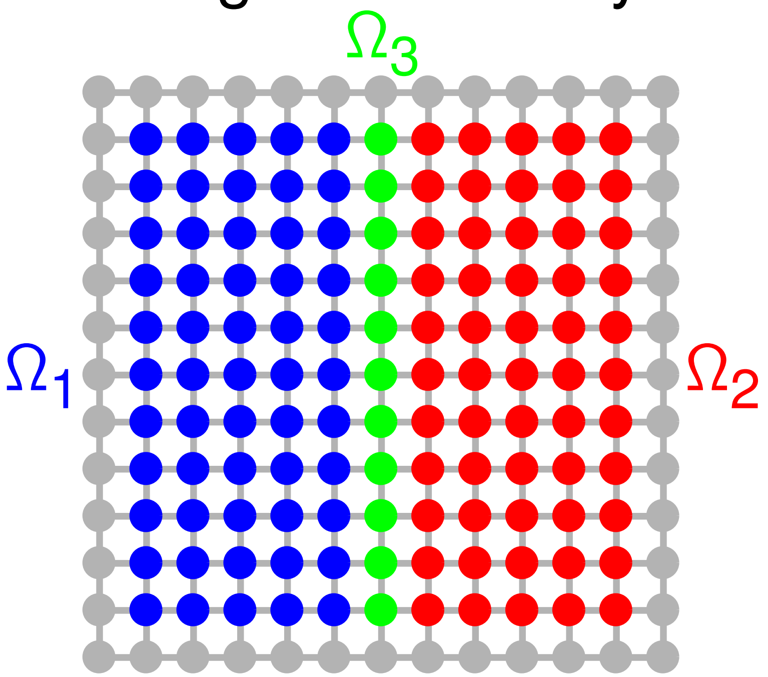
[2015, CAMWA, Hao/M./Young]

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



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

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



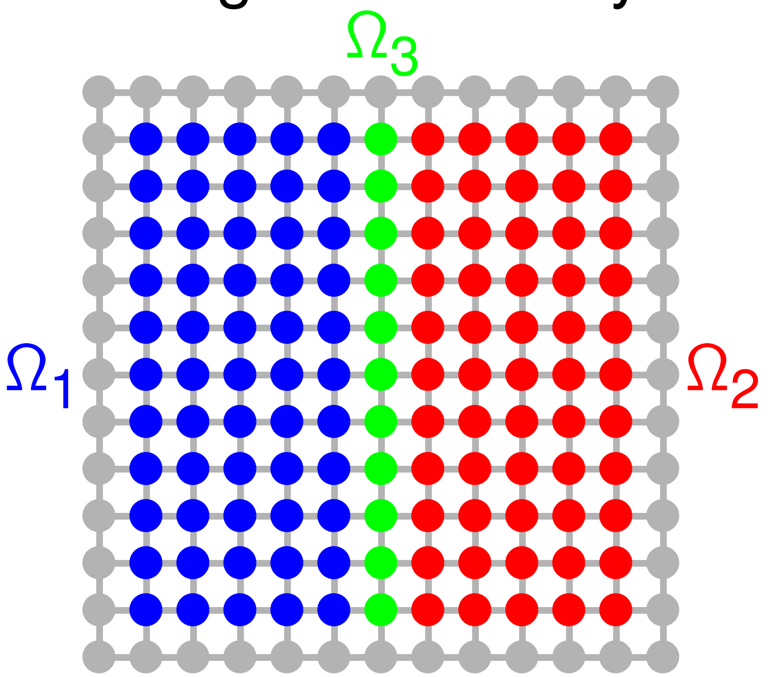
$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{array}\right]$$

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

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \hline \mathbf{A}_{31}\mathbf{A}_{11}^{-1} & \mathbf{A}_{32}\mathbf{A}_{22}^{-1} & \mathbf{I} \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{S}_{33} \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{I} & \mathbf{0} & \mathbf{A}_{11}^{-1}\mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{A}_{22}^{-1}\mathbf{A}_{23} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I} \end{array}\right]$$

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

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



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

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

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \hline \mathbf{A}_{31}\mathbf{A}_{11}^{-1} & \mathbf{A}_{32}\mathbf{A}_{22}^{-1} & \mathbf{I} \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{S}_{33} \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{I} & \mathbf{0} & \mathbf{A}_{11}^{-1}\mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{A}_{22}^{-1}\mathbf{A}_{23} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I} \end{array}\right]$$

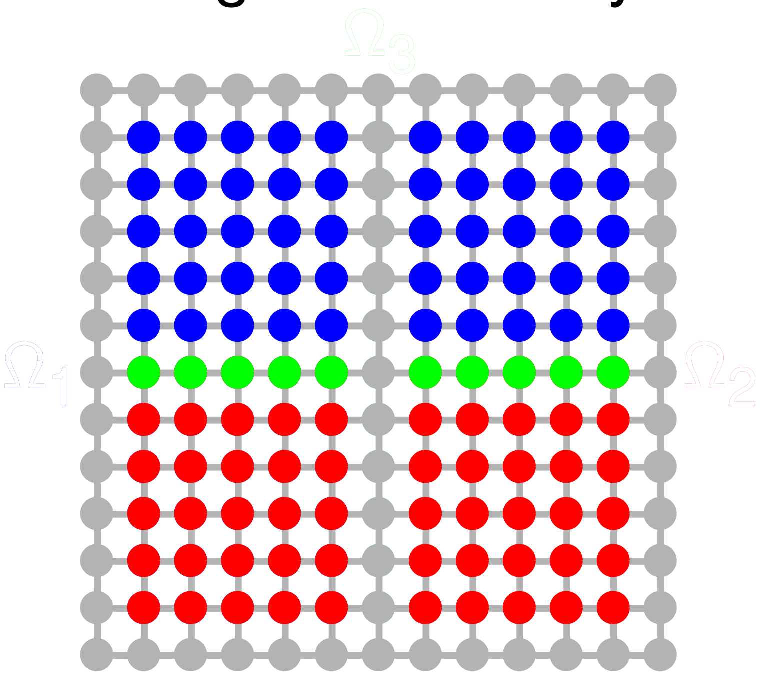
where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement.*

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

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

Notice the obvious recursion!

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



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

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

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \hline \mathbf{A}_{31}\mathbf{A}_{11}^{-1} & \mathbf{A}_{32}\mathbf{A}_{22}^{-1} & \mathbf{I} \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{S}_{33} \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{I} & \mathbf{0} & \mathbf{A}_{11}^{-1}\mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{A}_{22}^{-1}\mathbf{A}_{23} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I} \end{array}\right]$$

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

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

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

Notice the obvious recursion!

Example: Schur complements

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

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

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

Example: Schur complements

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

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

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

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

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

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

Example: Schur complements

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

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

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

Nested dissection solvers with $O(N)$ complexity — *Le Borne, Grasedyck, & Kriemann (2007)*, *Martinsson (2009)*, *J. Xia, Chandrasekaran, Gu, & Li (2009)*, *Gillman & Martinsson (2011)*, *Schmitz & L. Ying (2012)*, *Darve & Ambikasaran (2013)*, *Ho & Ying (2015)*, *Amestoy, Ashcraft, et al (2015)*, *Oseledets & Suchnikova (2015)*, etc.
$O(N)$ direct solvers for integral equations were developed by *Martinsson & Rokhlin (2005)*, *Greengard, Gueyffier, Martinsson, & Rokhlin (2009)*, *Gillman, Young, & Martinsson (2012)*, *Ho & Greengard (2012)*, *Ho & Ying (2015)*. Work in 1990's *Y. Chen, P. Starr, V. Rokhlin, L. Greengard, E. Michielssen.* Related to work on $\mathcal{H}$ and $\mathcal{H}^2$ matrix methods (1998 and forwards) by *Börm, Bebendorf, Hackbusch, Khoromskij, Sauter, etc.*

Example: Schur complements

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

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

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

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

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

Outline of talk

(1) **The role of global operators in scientific computing.**

(2) **Interaction ranks – why are they small?**

(3) **Introduction to rank structured matrices.**

(4) **Randomized method for compressing global operators.**

Interaction ranks: Why are they small?

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

Why is this the case?

Interaction ranks: Why are they small?

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

Why is this the case?

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

This effect has many well known physical consequences:

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

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

Interaction ranks: Boundary integral equations

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

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

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

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

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

The kernels are derived from the corresponding fundamental solutions:

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

where, as before,

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

Interaction ranks: Boundary integral equations

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

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

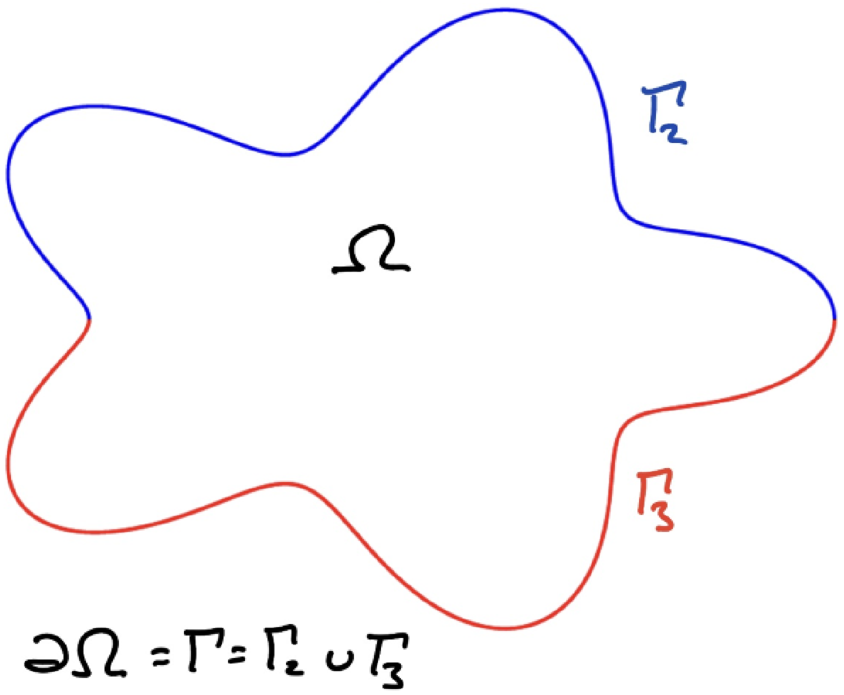
$$\alpha\sigma(\boldsymbol{x}) + \int_\Gamma \big(d(\boldsymbol{x},\boldsymbol{y}) + s(\boldsymbol{x},\boldsymbol{y})\big)\,\sigma(\boldsymbol{y})\,ds(\boldsymbol{y}) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma.$$

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

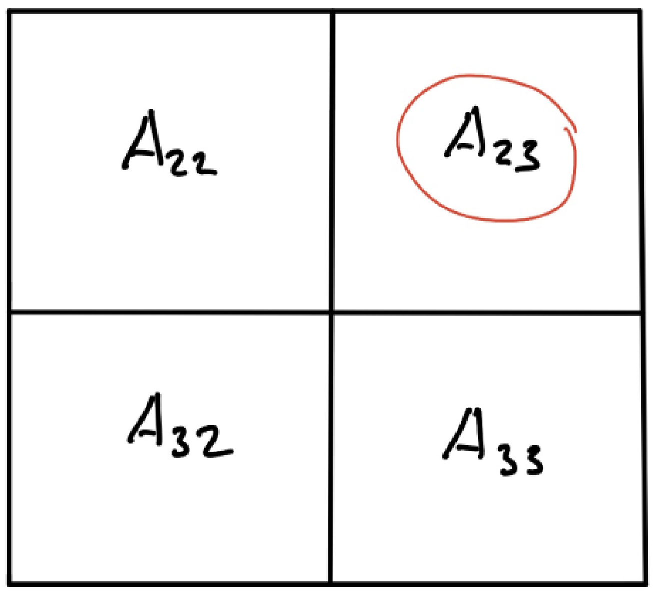
$$\beta\sigma(\boldsymbol{x}) + \int_\Gamma \big(d_\kappa(\boldsymbol{x},\boldsymbol{y}) + i\kappa s_\kappa(\boldsymbol{x},\boldsymbol{y})\big)\,\sigma(\boldsymbol{y})\,ds(\boldsymbol{y}) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma.$$

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

The geometry



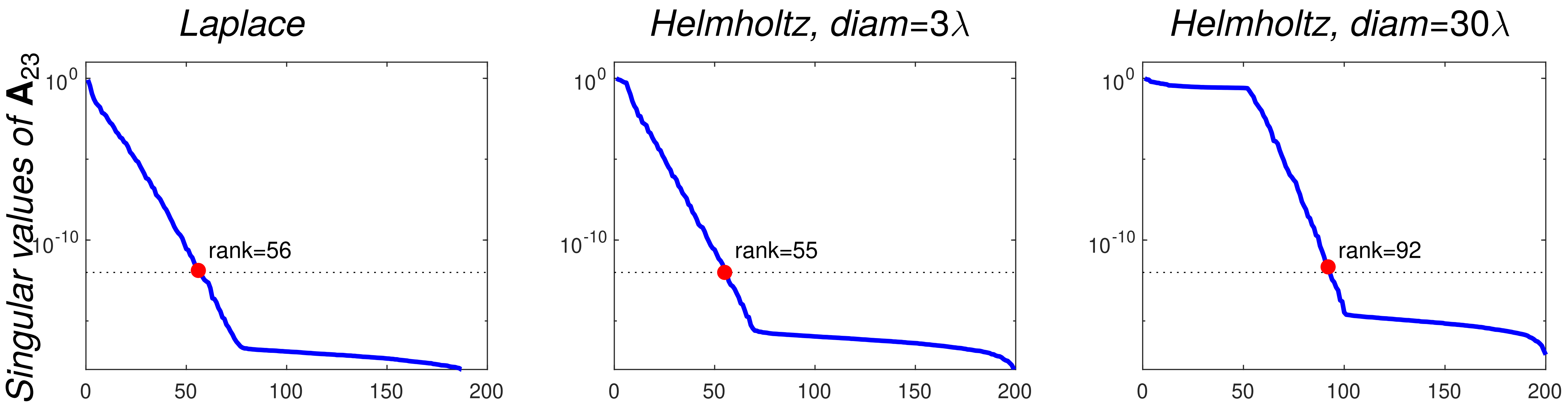
The matrix **A**



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

Interaction ranks: Boundary integral equations

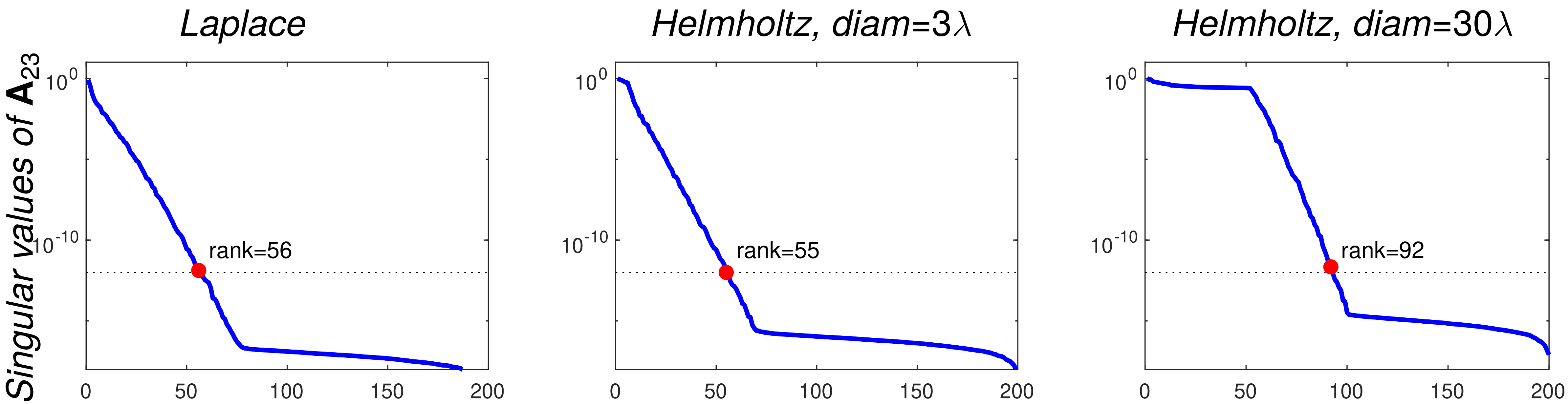
The ranks of an off-diagonal block of **A**:



This is all as expected. Somewhat accessible by analysis.

Interaction ranks: Boundary integral equations

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



Laplace

Helmholtz, diam=3λ

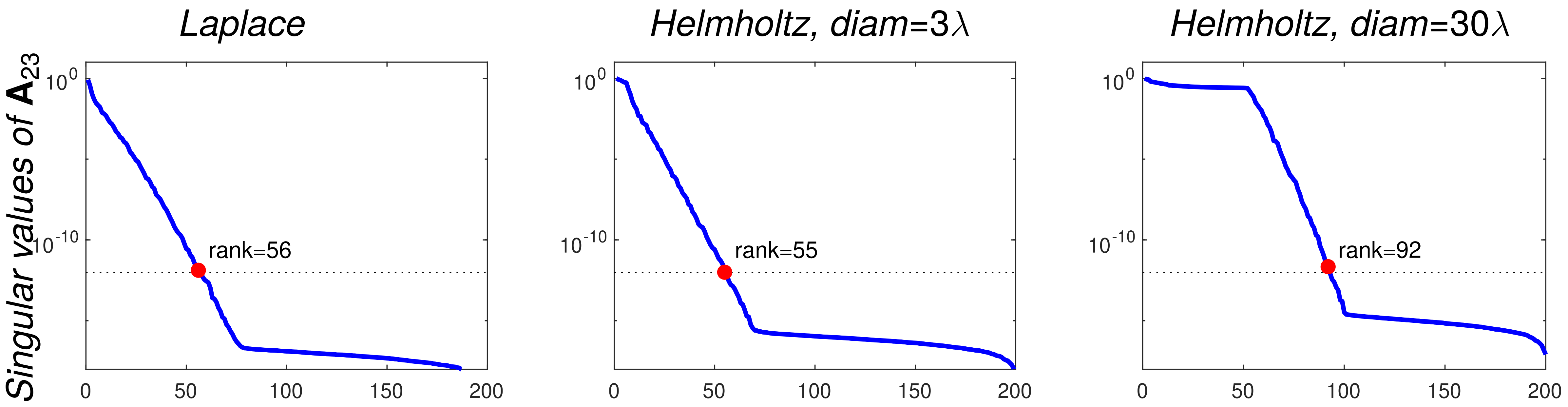
Helmholtz, diam=30λ

This is all as expected. Somewhat accessible by analysis.

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

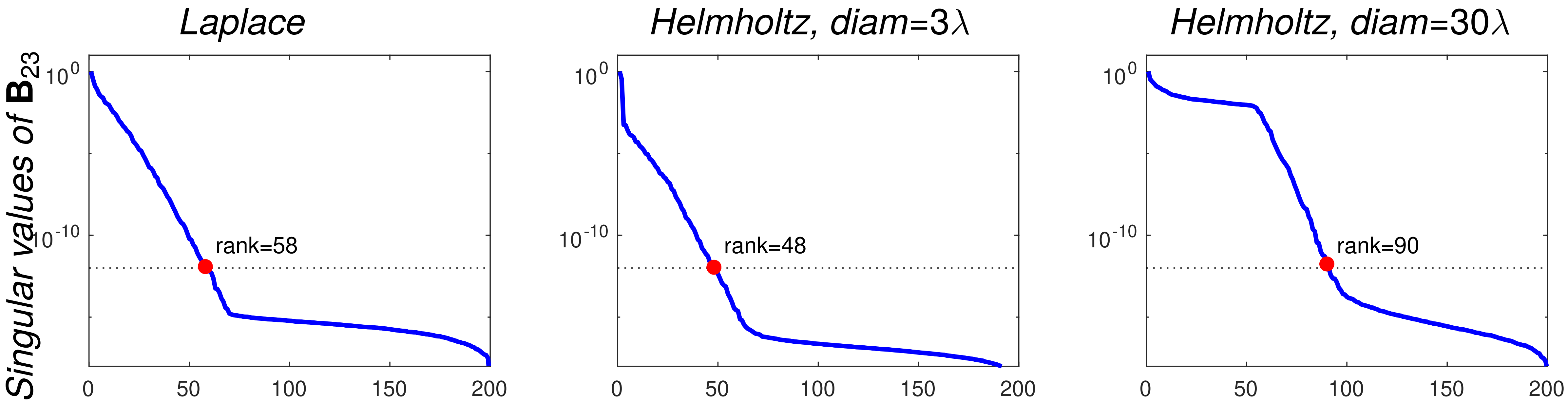
Interaction ranks: Boundary integral equations

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



This is all as expected. Somewhat accessible by analysis.

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

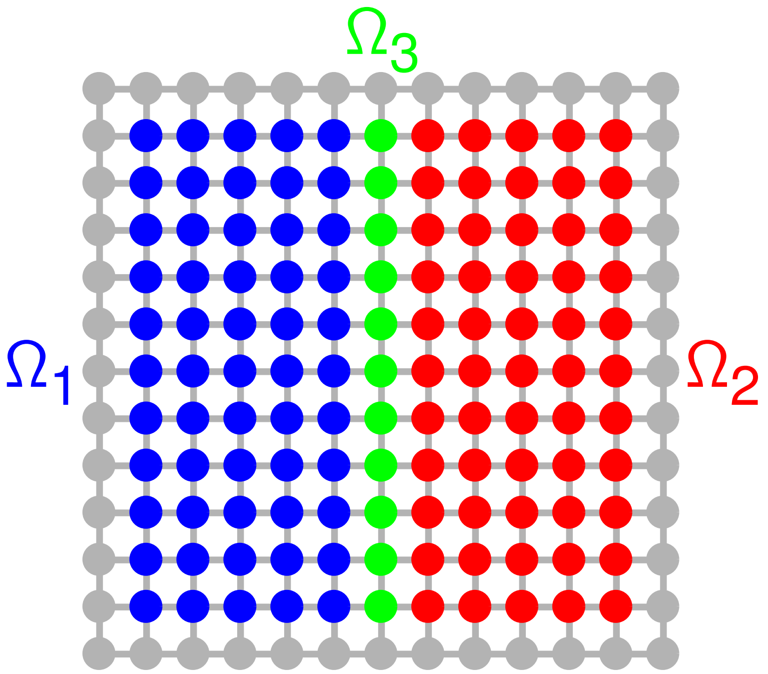


Remarkable similarity!

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

Interaction ranks: Stiffness matrix from finite difference discretization

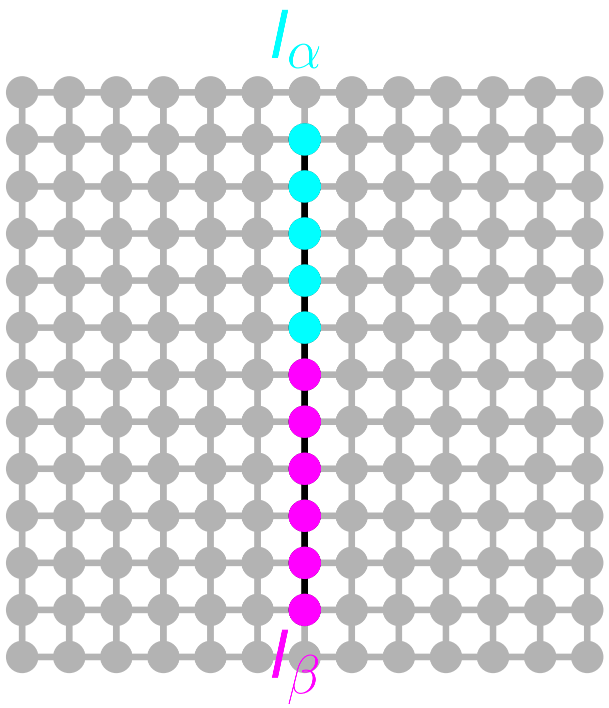
Recall our example of Laplace’s equation discretized using the 5-point stencil.



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

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

Then split the Schur complement into four parts:

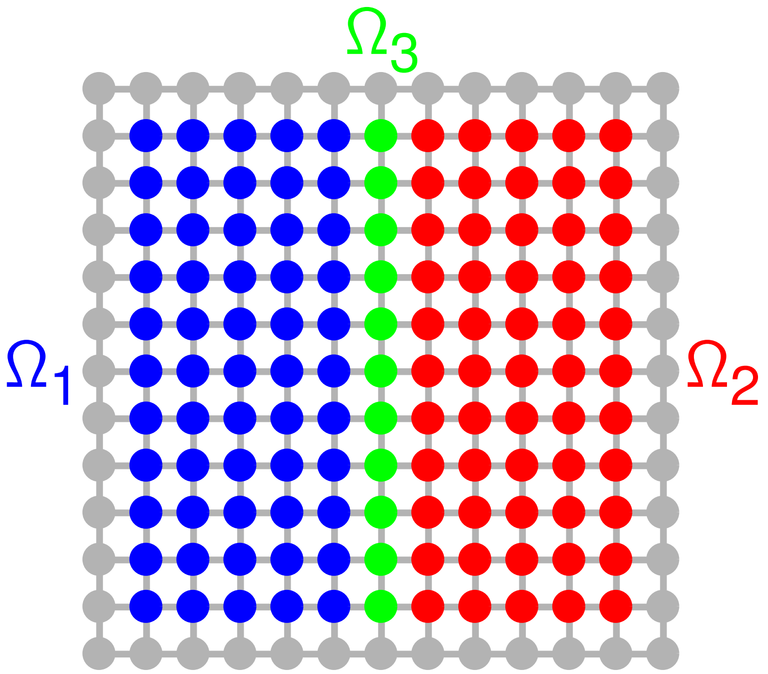


$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{\alpha\alpha} & \mathbf{S}_{\alpha\beta} \\ \mathbf{S}_{\beta\alpha} & \mathbf{S}_{\beta\beta} \end{bmatrix}$$

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

Interaction ranks: Stiffness matrix from finite difference discretization

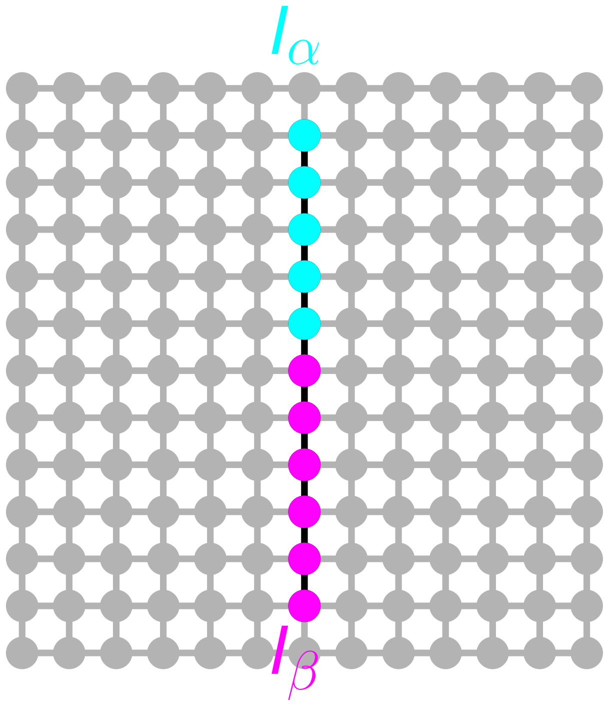
Recall our example of Laplace's equation discretized using the 5-point stencil.



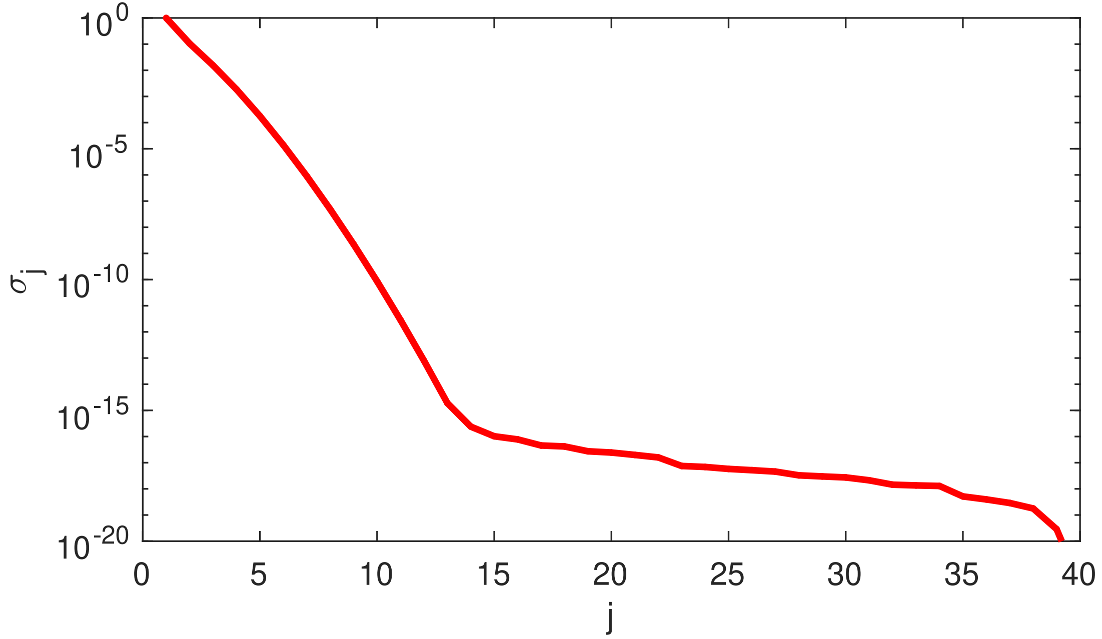
$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \hline \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{array}\right]$$

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

Then split the Schur complement into four parts:



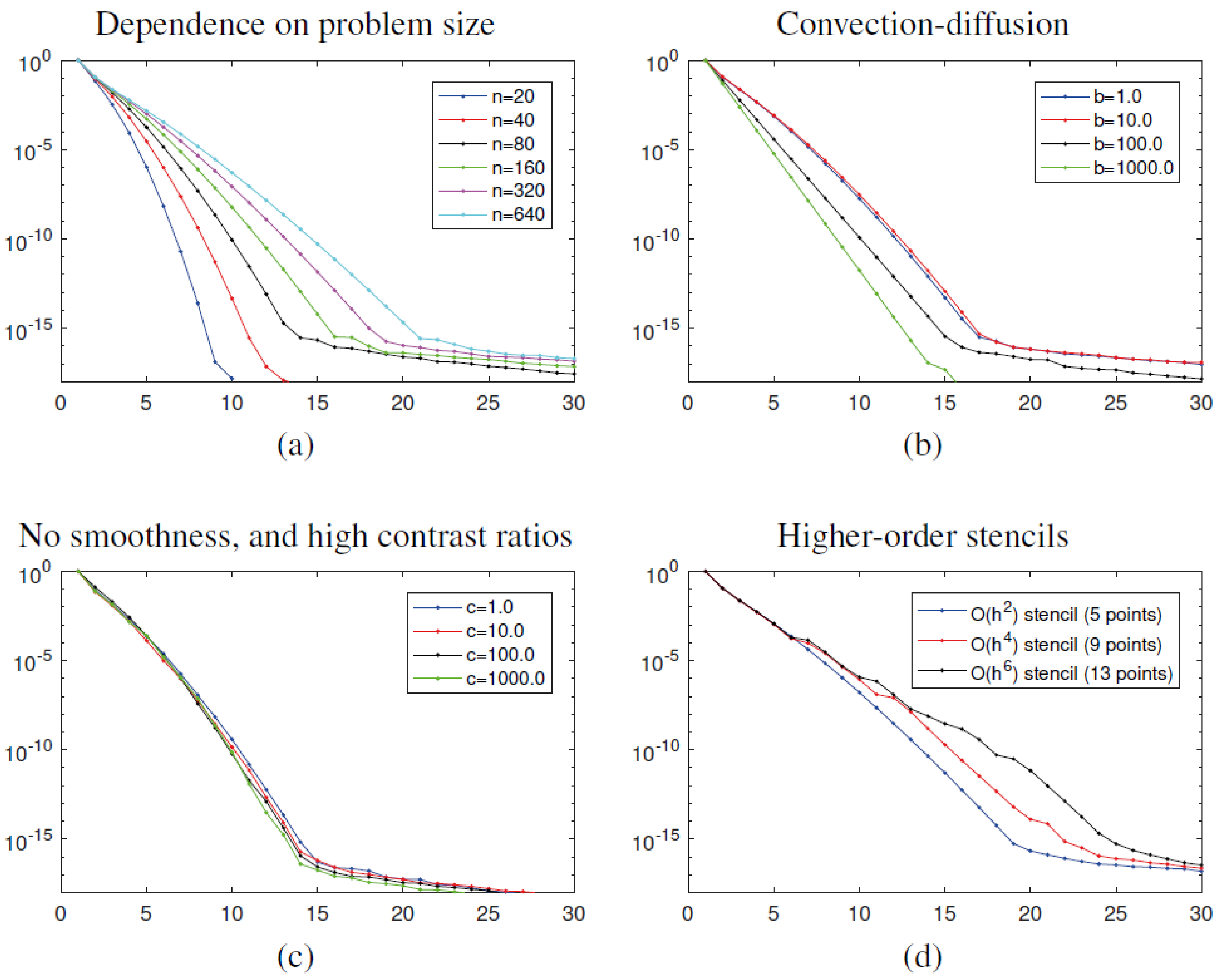
Singular values of $S_{\alpha\beta}$ for an 80×80 *grid.*



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

Interaction ranks: Stiffness matrix from finite difference discretization

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


Dependence on problem size

(a)

Convection-diffusion

(b)

No smoothness, and high contrast ratios

(c)

Higher-order stencils

(d)

Note: The rank decay property is remarkably stable!

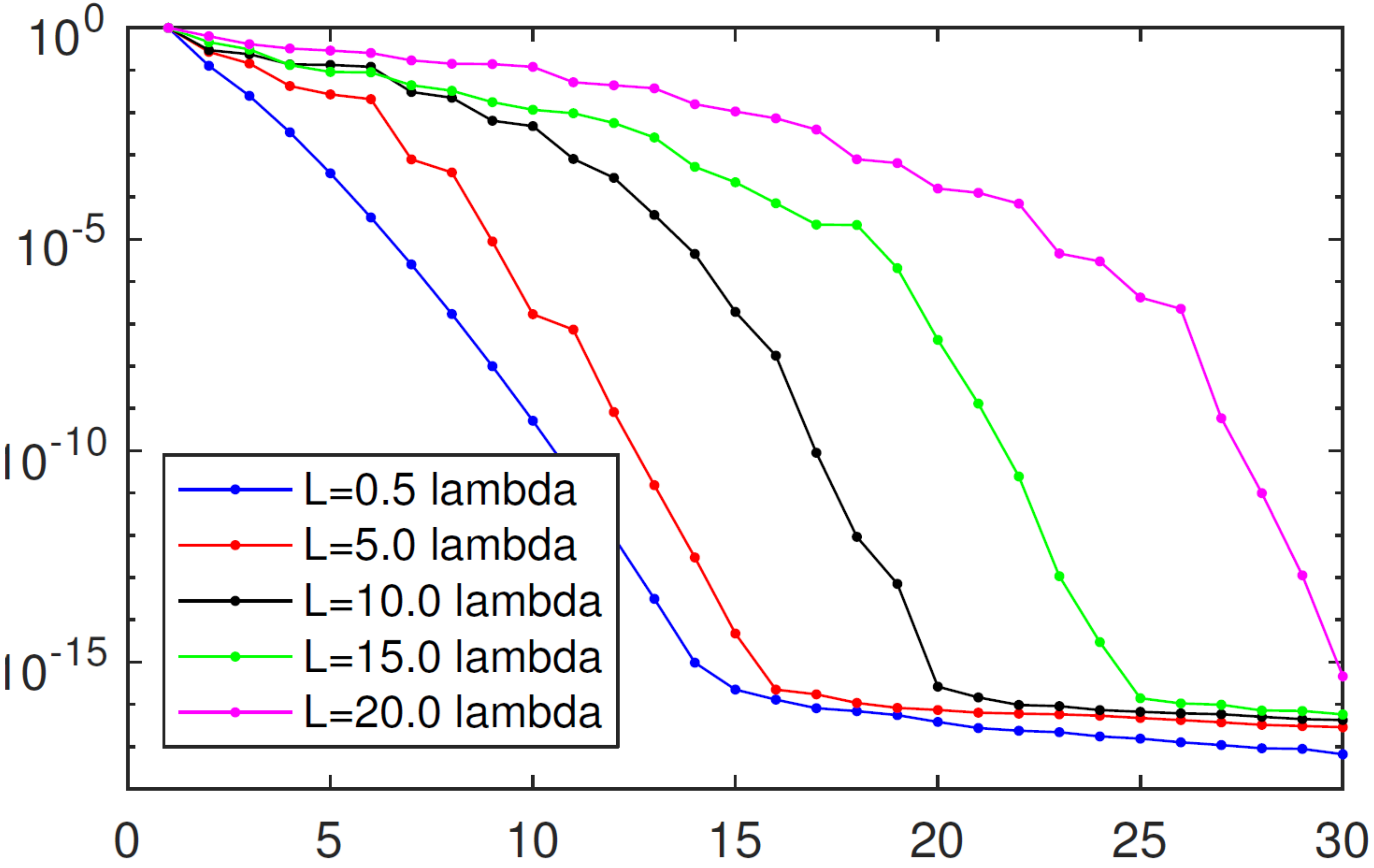
Note: The decay continues to $\epsilon_{\rm mach}$ — regardless of the discretization errors!

Interaction ranks: Stiffness matrix from finite difference discretization

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

Interaction ranks: Stiffness matrix from finite difference discretization

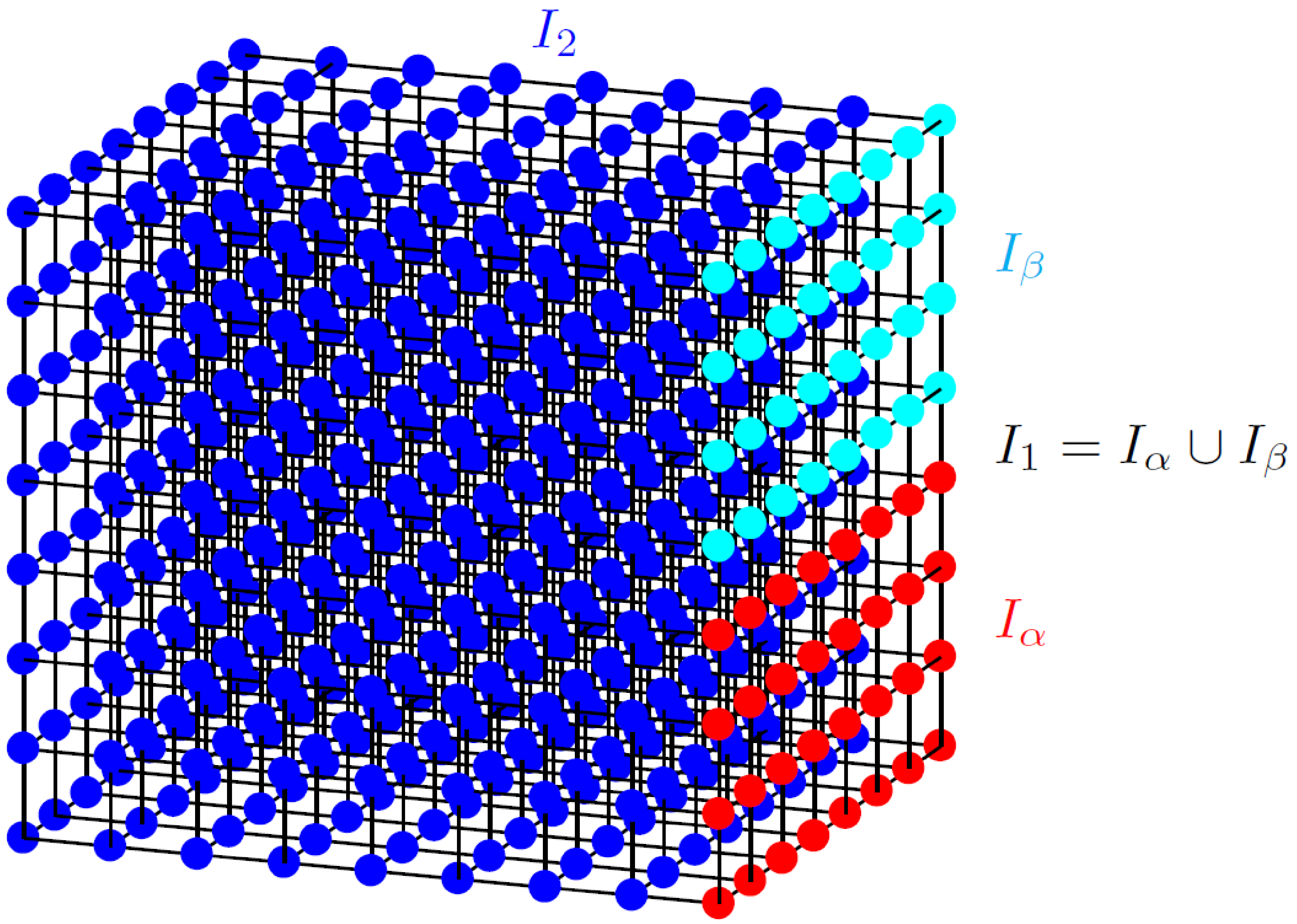
Next, let us consider Helmholtz problems with increasing wave numbers.



Fast decay *once oscillations are resolved.*

Interaction ranks: Stiffness matrix from finite difference discretization

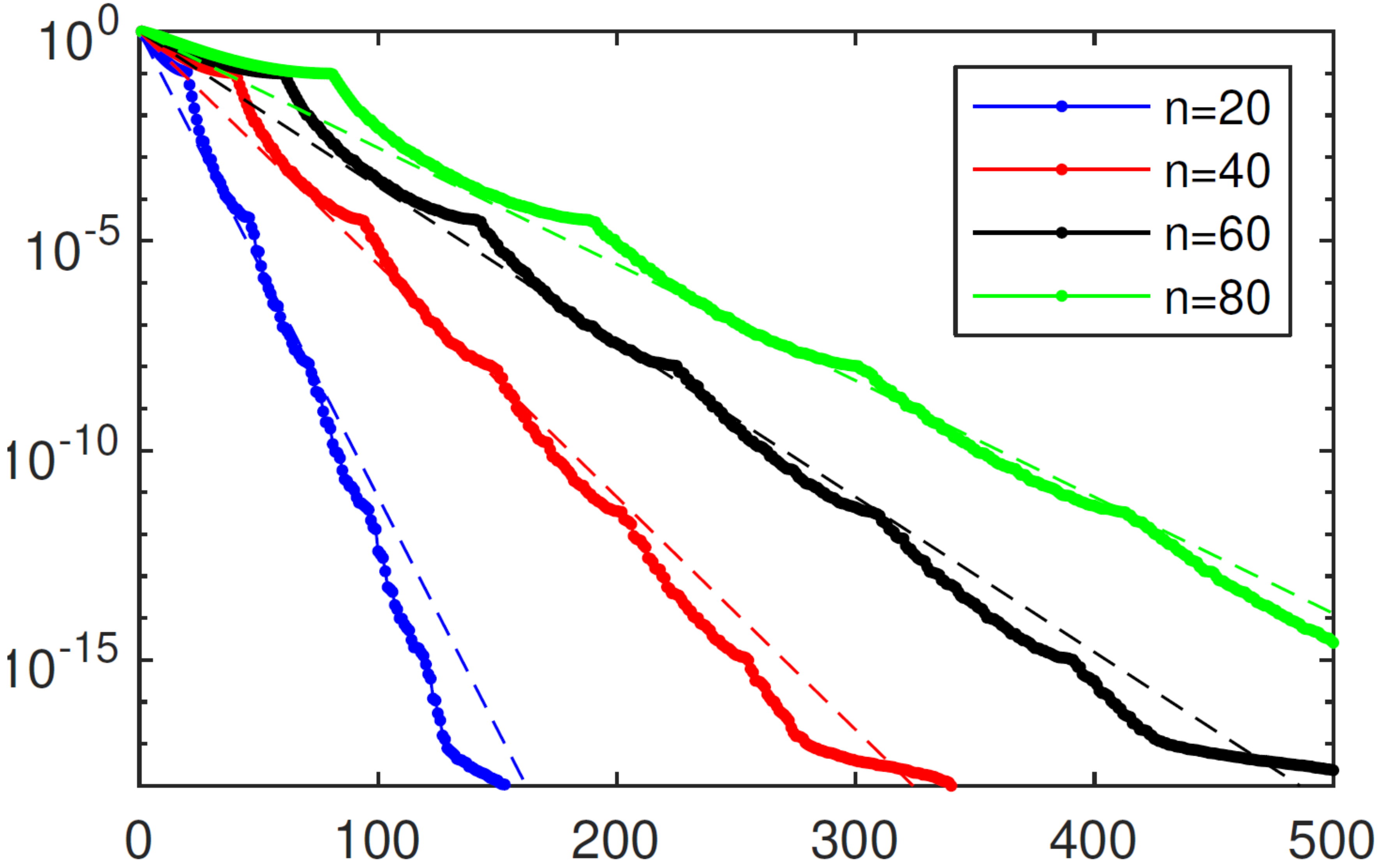
Finally, let us consider the analogous 3D problem.



The geometry.

Interaction ranks: Stiffness matrix from finite difference discretization

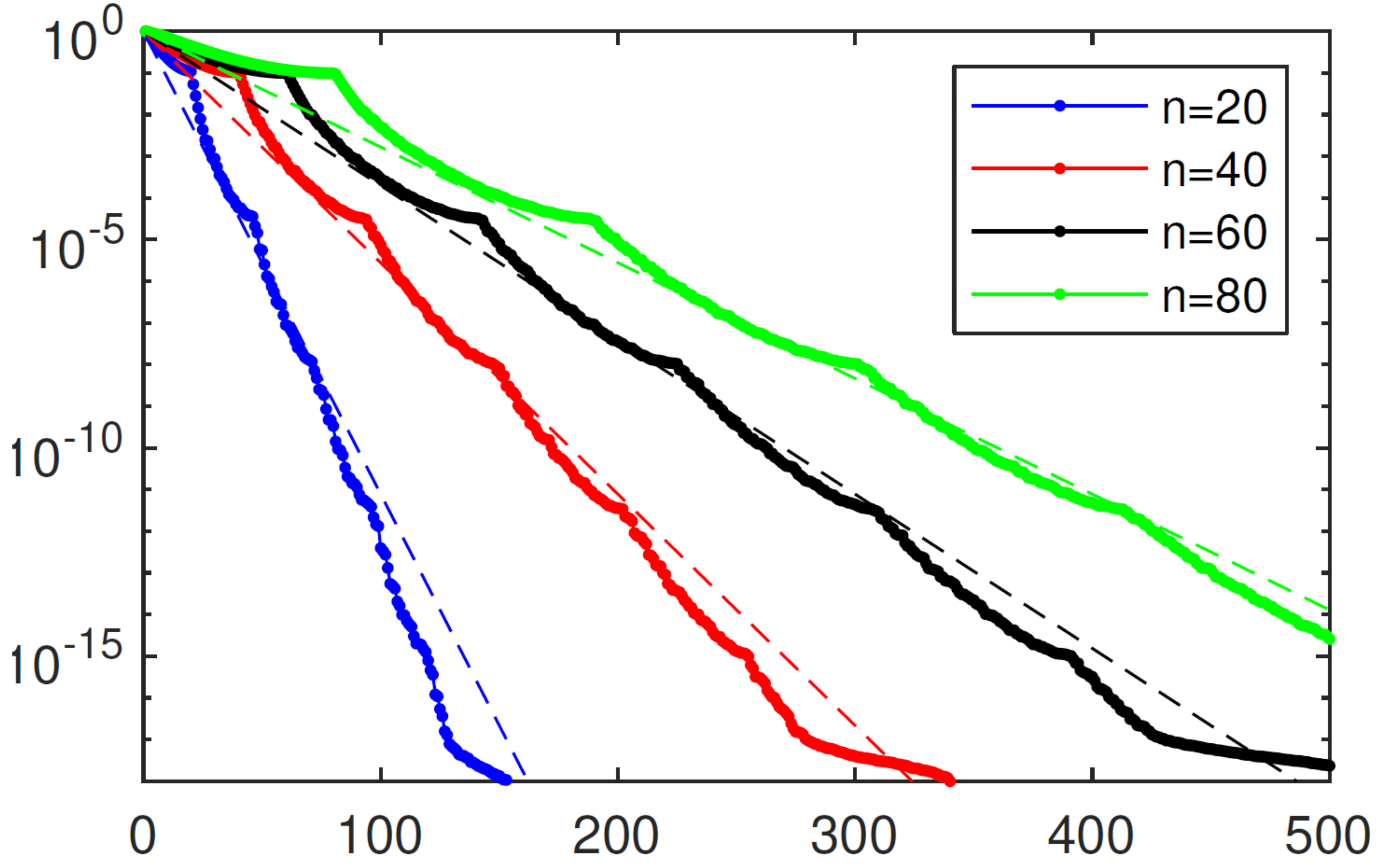
Finally, let us consider the analogous 3D problem.



The singular values.

Interaction ranks: Stiffness matrix from finite difference discretization

Finally, let us consider the analogous 3D problem.



The singular values.

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

Outline of talk

(1) **The role of global operators in scientific computing.**

(2) **Interaction ranks – why are they small?**

(3) **Introduction to rank structured matrices.**

(4) **Randomized method for compressing global operators.**

The Fast Multipole Method

A classical method that exploits the rank deficiency of long range interaction is the Fast Multipole Method, which is an $O(N)$ algorithm for evaluating sums of the form

$$\begin{array}{cccc} \mathbf{u} & = & \mathbf{A} & \mathbf{q} \\ N \times 1 & & N \times N & N \times 1 \end{array}$$

or, equivalently,

$$u_i = \sum_{j=1}^{J} \mathbf{A}(i,j)\, q_j, \qquad i = 1, 2, \dots, N,$$

where $\mathbf{q}$ is a vector of *sources*, where $\mathbf{u}$ is a vector of *potentials*, and where $\mathbf{A}$ is a (dense) kernel matrix of the form

$$\mathbf{A}(i,j) = \phi(\boldsymbol{x}_i - \boldsymbol{x}_j)$$

for some set of source locations $\{\boldsymbol{x}_i\}_{i=1}^{N}$. The function ϕ is a fundamental solution of one of the standard elliptic PDEs of mathematical physics. For instance (2D Laplace),

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

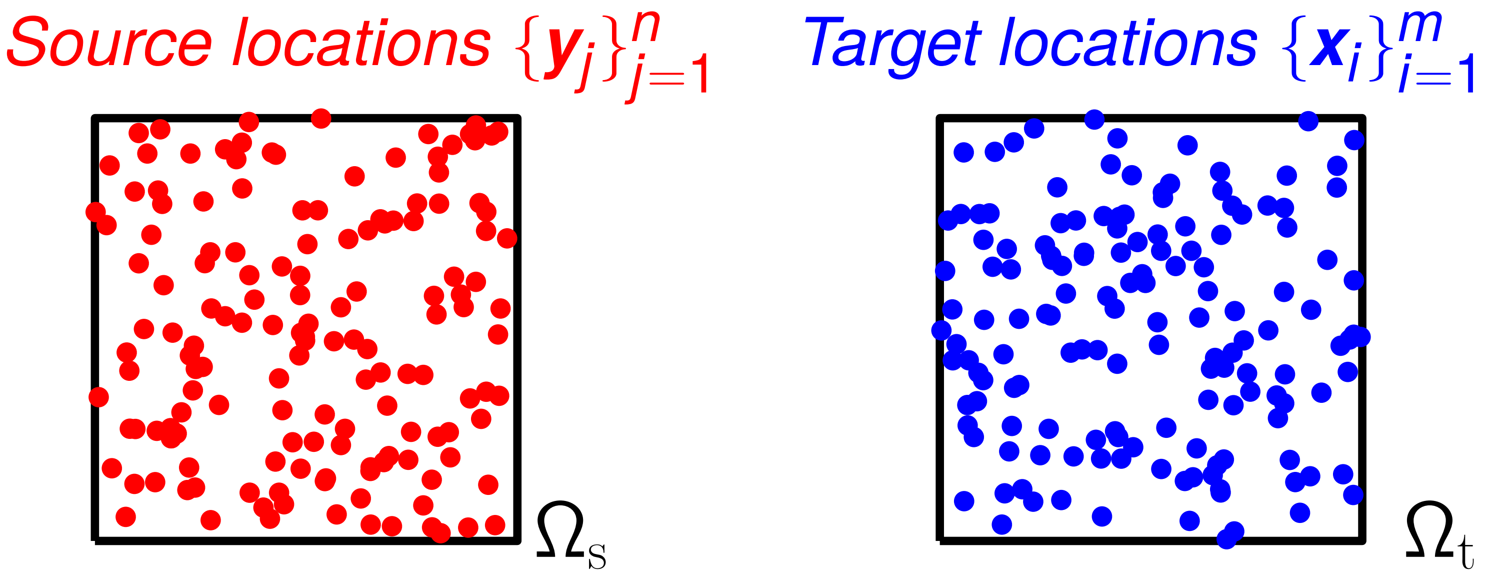
The key to the FMM is that the function ϕ is to high precision *separable* when the source points and the target points are well separated.

The Fast Multipole Method

Illustration: Consider a simplified summation problem

$$u_i = \sum_{j=1}^{n} \phi(\boldsymbol{x}_i - \boldsymbol{y}_j)\, q_j, \qquad i = 1, 2, \dots, m$$

where the source locations $\{\boldsymbol{y}_j\}_{j=1}^{n}$ are well separated from the target locations $\{\boldsymbol{x}_i\}_{i=1}^{m}$:


Source locations $\{\boldsymbol{y}_j\}_{j=1}^{n}$
Target locations $\{\boldsymbol{x}_i\}_{i=1}^{m}$
$\Omega_{\rm s}$
$\Omega_{\rm t}$

A classical multipole expansion shows that the kernel is approximately separable:

$$\phi(\boldsymbol{x} - \boldsymbol{y}) = \sum_{p=1}^{P} B_p(\boldsymbol{x})\, C_p(\boldsymbol{y}) + O\left((\sqrt{2}/3)^P\right)$$

This means that the sum can be evaluated in $\sim P(M + N)$ flops rather than $\sim MN$ flops.

(Multiplication by a matrix of rank P, instead of multiplication by a dense matrix.)

The Fast Multipole Method

For the “real” problem where the sources and the target sets are the same, a quadtree of boxes is introduced, and then a hierarchical algorithm is deployed:

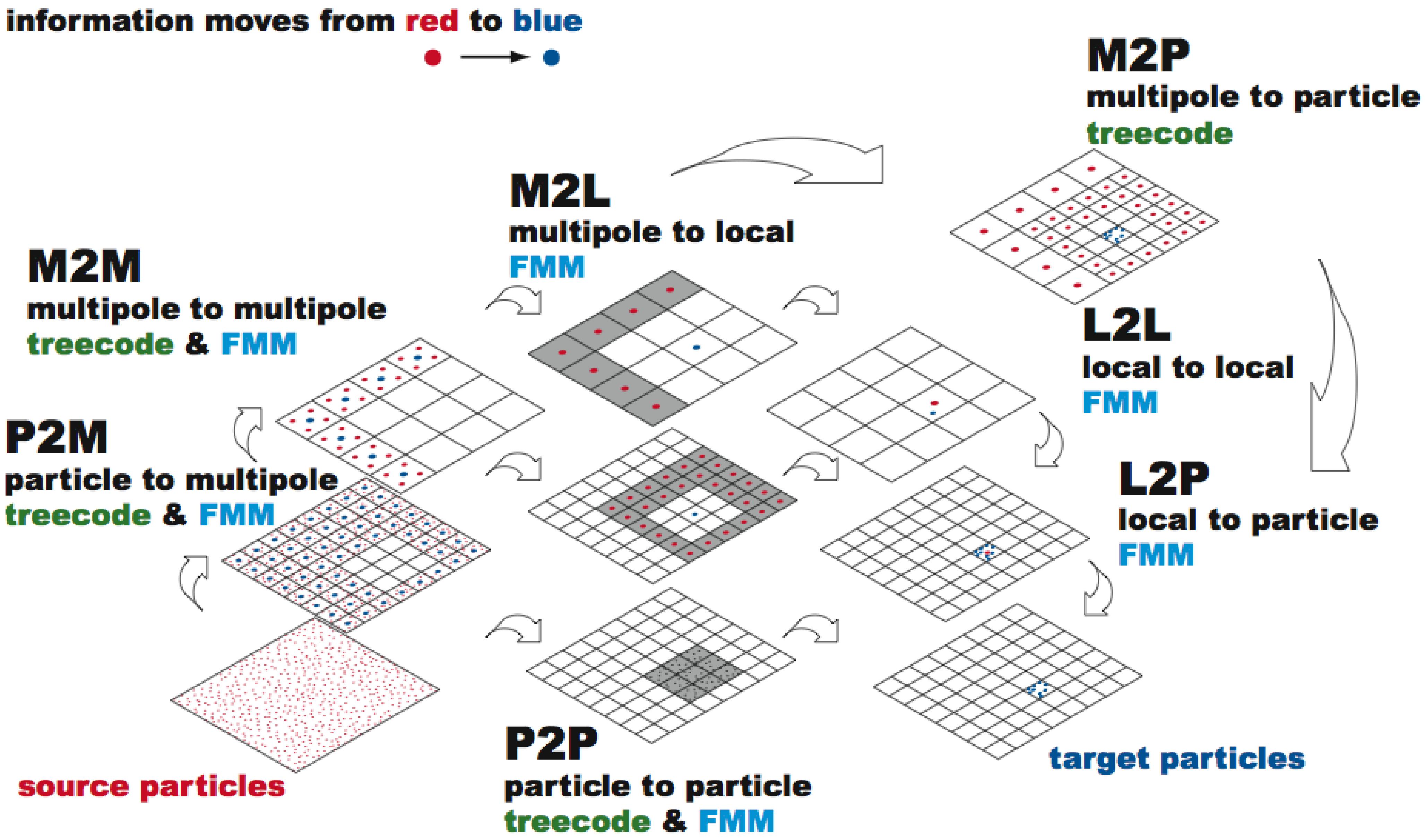


Image credit: Rio Yokota — https://www.bu.edu/pasi/courses/12-steps-to-having-a-fast-multipole-method-on-gpus/

Instead of describing the details of the FMM, let us tell the story from the point of view of algebra of rank structured matrices, starting from the most basic problems.

Inversion of structured matrices: Tridiagonal

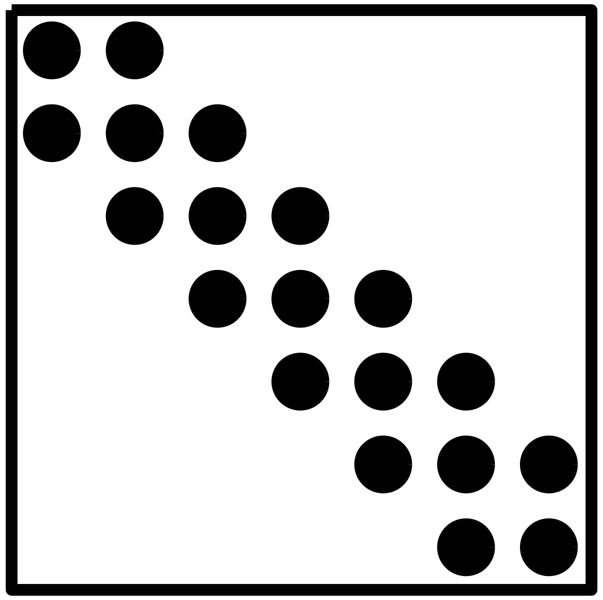
Consider a simple 2-point BVP on the interval $[0,1]$:

$$\text{(BVP)}\qquad \begin{cases} -u''(x) + p(x)\,u'(x) + m(x)\,u(x) = g(x), & x \in (0,1), \\ u(0) = f_{\rm L}, & \\ u(1) = f_{\rm R}. & \end{cases}$$

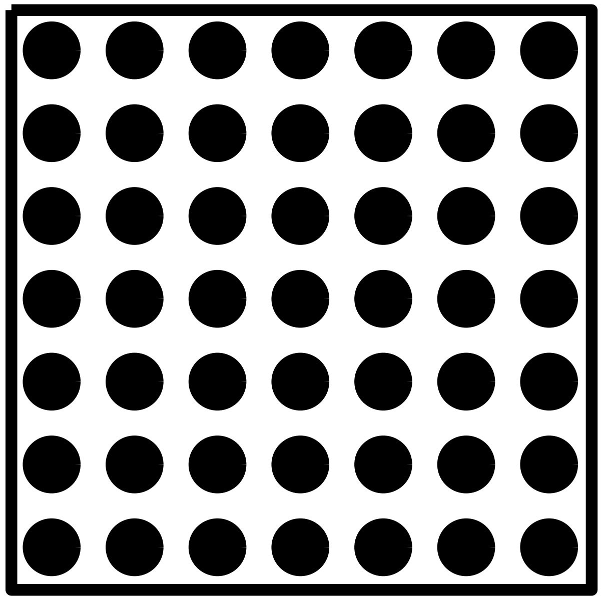
Discretizing (BVP) using a standard second order finite difference scheme, we get

$$\mathbf{Au} = \mathbf{b},$$

where $\mathbf{A}$ is a sparse matrix of size, say, $n \times n$. Then $\mathbf{A}^{-1}$ is dense.



Sparsity pattern of **A***.*



Sparsity pattern of $\mathbf{A}^{-1}$*.*

Inversion of structured matrices: Tridiagonal

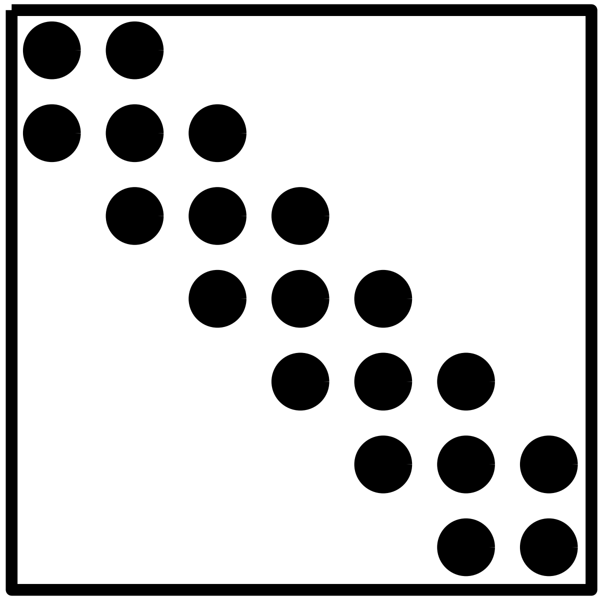
Consider a simple 2-point BVP on the interval $[0,1]$:

$$\text{(BVP)} \qquad \begin{cases} -u''(x) + p(x)\,u'(x) + m(x)\,u(x) = g(x), & x \in (0,1), \\ u(0) = f_{\mathrm{L}}, & \\ u(1) = f_{\mathrm{R}}. & \end{cases}$$

Discretizing (BVP) using a standard second order finite difference scheme, we get

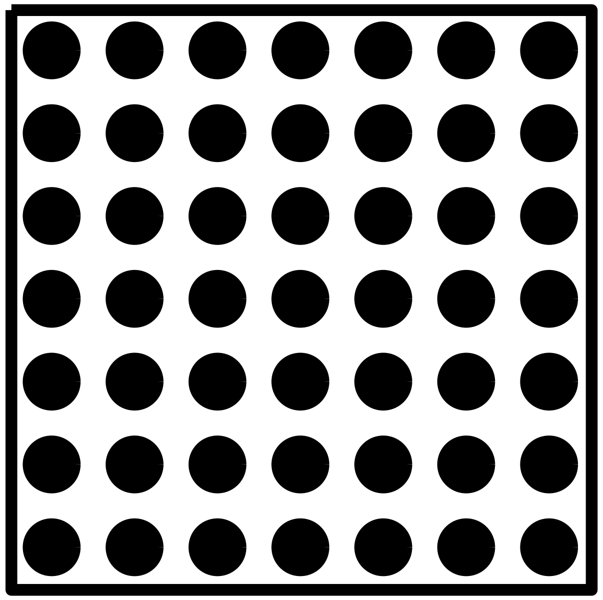
$$\mathbf{Au} = \mathbf{b},$$

where $\mathbf{A}$ is a sparse matrix of size, say, $n \times n$. Then $\mathbf{A}^{-1}$ is dense.



Sparsity pattern of $\mathbf{A}$*.*

$\mathbf{A}$ *is tridiagonal.*



Sparsity pattern of $\mathbf{A}^{-1}$*.*

Inversion of structured matrices: Tridiagonal

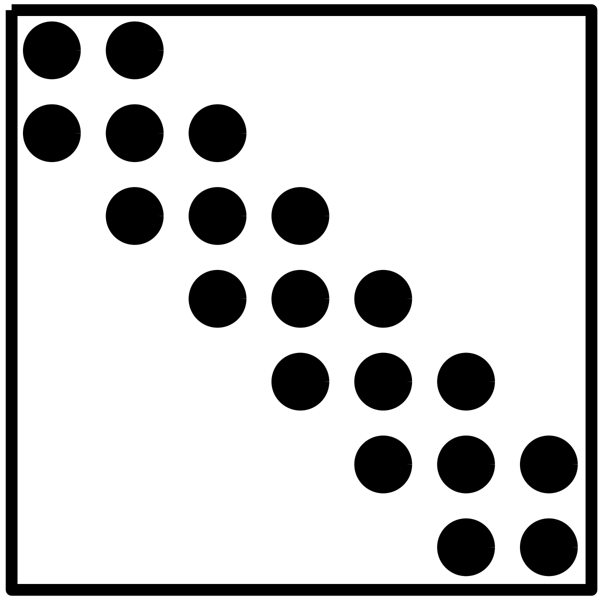
Consider a simple 2-point BVP on the interval $[0, 1]$:

$$\text{(BVP)} \qquad \begin{cases} -u''(x) + p(x)\,u'(x) + m(x)\,u(x) = g(x), & x \in (0,1), \\ u(0) = f_{\mathrm{L}}, & \\ u(1) = f_{\mathrm{R}}. & \end{cases}$$

Discretizing (BVP) using a standard second order finite difference scheme, we get

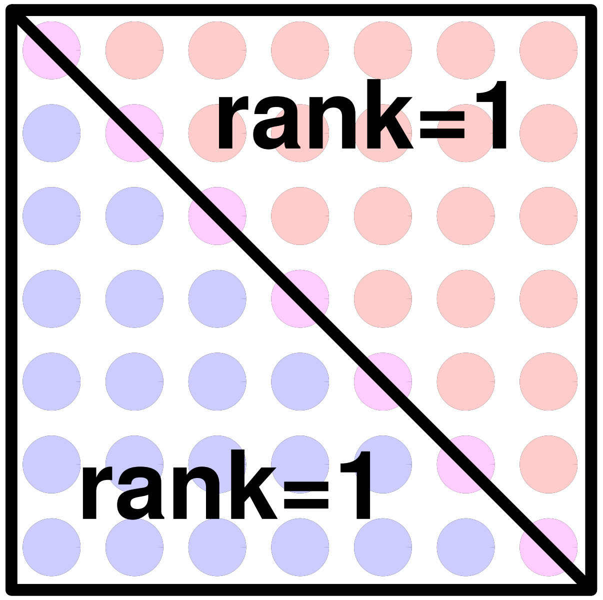
$$\mathbf{Au} = \mathbf{b},$$

where $\mathbf{A}$ is a sparse matrix of size, say, $n \times n$. Then $\mathbf{A}^{-1}$ is dense.



Sparsity pattern of $\mathbf{A}$.

$\mathbf{A}$ *is tridiagonal.*



Sparsity pattern of $\mathbf{A}^{-1}$.

$\mathbf{A}^{-1}$ *is semi-separable.*

Inversion of structured matrices: Tridiagonal

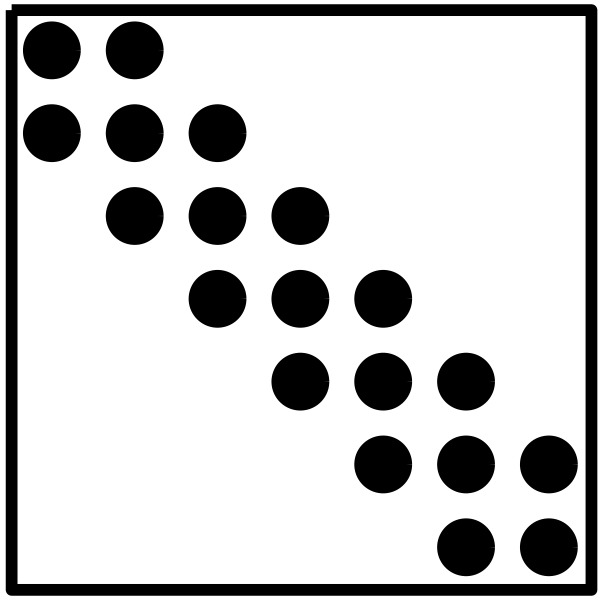
Consider a simple 2-point BVP on the interval $[0,1]$:

$$\text{(BVP)} \qquad \begin{cases} -u''(x) + p(x)\,u'(x) + m(x)\,u(x) = g(x), & x \in (0,1), \\ u(0) = f_{\mathrm{L}}, & \\ u(1) = f_{\mathrm{R}}. & \end{cases}$$

Discretizing (BVP) using a standard second order finite difference scheme, we get

$$\mathbf{A}\mathbf{u} = \mathbf{b},$$

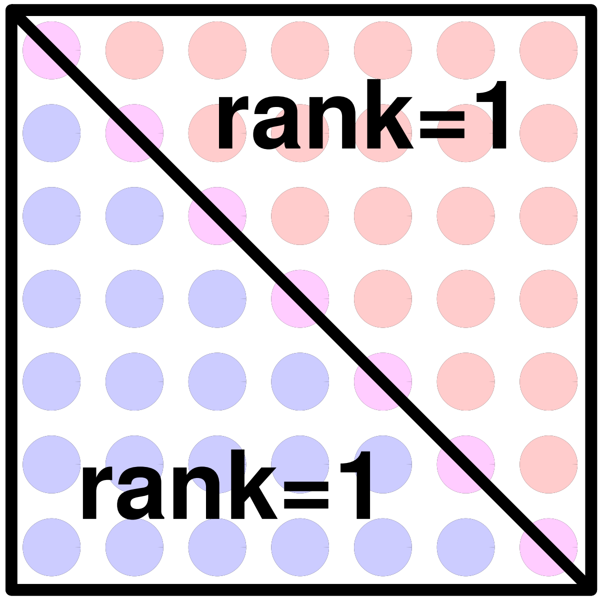
where $\mathbf{A}$ is a sparse matrix of size, say, $n \times n$. Then $\mathbf{A}^{-1}$ is dense.



Sparsity pattern of $\mathbf{A}$.

$\mathbf{A}$ *is tridiagonal.*

$\mathbf{A}$ *is sparse.*



Sparsity pattern of $\mathbf{A}^{-1}$.

$\mathbf{A}^{-1}$ *is semi-separable.*

$\mathbf{A}^{-1}$ *is data-sparse.*

Inversion of structured matrices: Tridiagonal

Consider a simple 2-point BVP on the interval $[0,1]$:

$$\text{(BVP)}\qquad \begin{cases} -u''(x)+p(x)\,u'(x)+m(x)\,u(x)=g(x), & x\in(0,1),\\ u(0)=f_{\mathrm{L}}, & \\ u(1)=f_{\mathrm{R}}. & \end{cases}$$

Discretizing (BVP) using a standard second order finite difference scheme, we get

$$\mathbf{Au}=\mathbf{b},$$

where $\mathbf{A}$ is a sparse matrix of size, say, $n\times n$. Then $\mathbf{A}^{-1}$ is dense.

Fun facts:

- If $\mathbf{A}$ is invertible and tridiagonal, then $\mathbf{A}^{-1}$ is semi-separable (meaning that the upper triangular and the lower triangular parts are restrictions of rank 1 matrices).
- If $\mathbf{A}$ is invertible and semi-separable, then $\mathbf{A}^{-1}$ is tridiagonal.
- Cost of storage is $3n-2$ floats in either case.
- Cost of inversion is $O(n)$ in either case.

Note: LU factorization is more common: Factors $\mathbf{L}$ and $\mathbf{U}$ are each bidiagonal.

Note: Without preconditioning, an iterative method needs *at least* $O(n)$ iterations.

Inversion of structured matrices: Semi-separable plus diagonal

Let us again consider a two point boundary value problem

$$\text{(BVP)} \qquad -u''(y) + m(y)\,u(y) = g(y), \qquad y \in (0,1),$$

now with zero boundary data. Recall that when $m = 0$, we can solve (BVP) analytically:

$$u(x) = \int_0^1 G(x,y)\,g(y)\,dy,$$

where the *Green's function* G (which is semi-separable!) takes the form

$$G(x,y) = \begin{cases} \dfrac{(b-x)(y-a)}{b-a}, & \text{when } x \geq y \qquad \text{(on or below the diagonal)}, \\ \dfrac{(x-a)(b-y)}{b-a}, & \text{when } x \leq y \qquad \text{(on or above the diagonal)}. \end{cases}$$

Multiply (BVP) by $G(x,y)$ and integrate in y over $[0,1]$ to get

$$u(x) + \int_0^1 G(x,y)\,m(y)\,u(y)\,dy = h(x), \qquad x \in [0,1],$$

where

$$h(x) = \int_0^1 G(x,y)\,g(y)\,dy.$$

Fact 1: The equation (BVP) is equivalent to

$$\text{(IE)} \qquad u(x) + \int_0^1 G(x,y)\,m(y)\,u(y)\,dy = h(x), \qquad x \in [0,1],$$

Inversion of structured matrices: Semi-separable plus diagonal

Let us again consider a two point boundary value problem

$$\text{(BVP)} \qquad -u''(y) + m(y)\,u(y) = g(y), \qquad y \in (0,1),$$

now with zero boundary data.

Fact 1: The equation (BVP) is equivalent to

$$\text{(IE)} \qquad u(x) + \int_0^1 G(x,y)\,m(y)\,u(y)\,dy = h(x), \qquad x \in [0,1],$$

where the *Green's function* (which is semi-separable!) takes the form

$$G(x,y) = \begin{cases} \dfrac{(b-x)(y-a)}{b-a}, & \text{when } x \geq y \qquad \text{(on or below the diagonal),} \\ \dfrac{(x-a)(b-y)}{b-a}, & \text{when } x \leq y \qquad \text{(on or above the diagonal).} \end{cases}$$

Inversion of structured matrices: Semi-separable plus diagonal

Let us again consider a two point boundary value problem

$$\text{(BVP)} \qquad -u''(y) + m(y)\,u(y) = g(y), \qquad y \in (0,1),$$

now with zero boundary data.

Fact 1: The equation (BVP) is equivalent to

$$\text{(IE)} \qquad u(x) + \int_0^1 G(x,y)\,m(y)\,u(y)\,dy = h(x), \qquad x \in [0,1],$$

where the *Green's function* (which is semi-separable!) takes the form

$$G(x,y) = \begin{cases} \dfrac{(b-x)(y-a)}{b-a}, & \text{when } x \geq y \qquad \text{(on or below the diagonal)},\\ \dfrac{(x-a)(b-y)}{b-a}, & \text{when } x \leq y \qquad \text{(on or above the diagonal)}. \end{cases}$$

Fact 2: Discretizing (IE) using a Nyström method with a uniform grid $\{x_i\}_{i=0}^{n+1} \subset [0,1]$ and the basic Trapezoidal rule results in the linear system

$$\big(\mathbf{I} + \mathbf{G}\mathbf{M}\big)\mathbf{u} = \mathbf{G}\mathbf{g}.$$

The $n \times n$ matrix $\mathbf{G}$ is *semi-separable* since it has entries

$$\mathbf{G}(i,j) = h\,G(x_i, x_j).$$

The matrix $\mathbf{M}$ is the diagonal matrix whose diagonal entries are $\{m(x_i)\}_{i=1}^{n}$.

Inversion of structured matrices: Semi-separable plus diagonal

Observe that we introduced two different approaches for solving 2 point BVPs:

- Finite difference (FD) discretization $\rightarrow$ *sparse* linear system.
- Integral equation (IE) discretization $\rightarrow$ *dense* linear system.

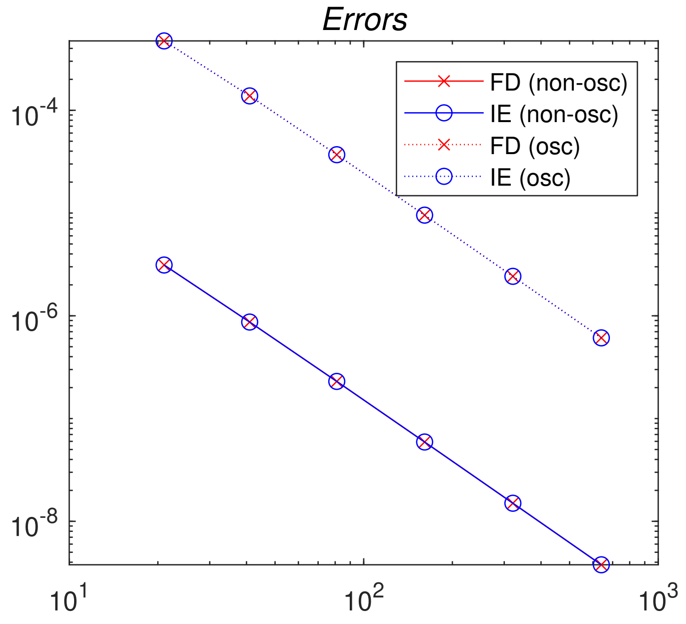
How do the two methods compare?

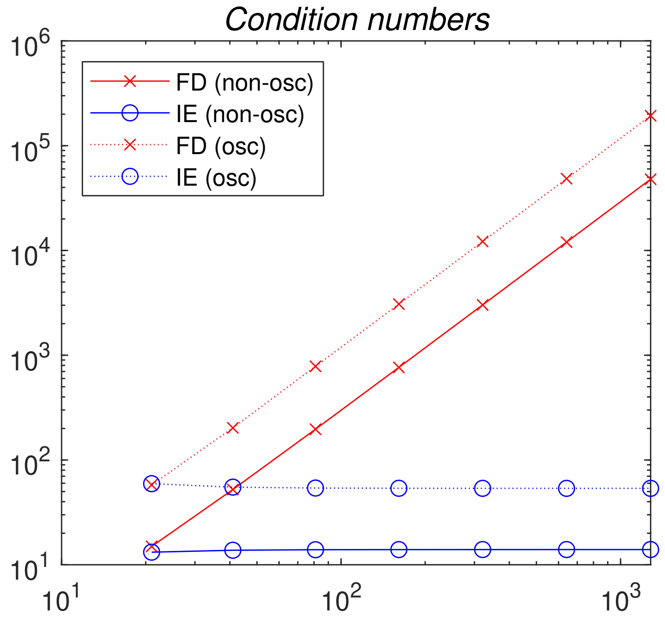
Inversion of structured matrices: Semi-separable plus diagonal

Observe that we introduced two different approaches for solving 2 point BVPs:

- Finite difference (FD) discretization → *sparse* linear system.
- Integral equation (IE) discretization → *dense* linear system.

How do the two methods compare?





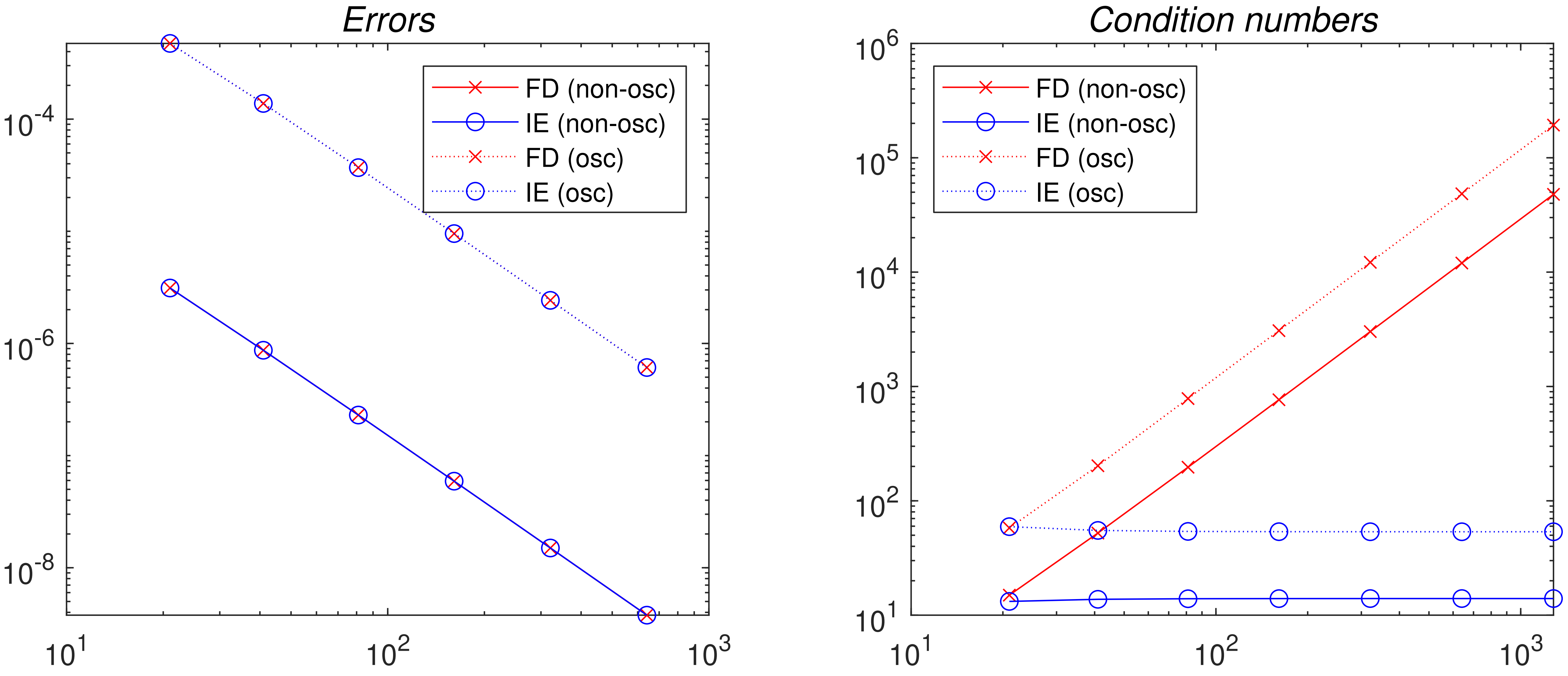
We considered a non-oscillatory problem “(non-osc)” where $m(x) = 100(1 + x)\cos(x)$ and $g(x) = 1 + \cos(1 + x)$. We then swapped the sign of m (but kept everything else the same) to get a problem with an oscillatory solution “(osc)”. The left plot shows the errors incurred (in max norm), while the right one shows the condition numbers of the coefficient matrices. The key point here is that the condition numbers of the integral equation formulation does not grow with n. A secondary point is to show that elliptic problems with oscillatory solutions are far more challenging.

Inversion of structured matrices: Semi-separable plus diagonal

Observe that we introduced two different approaches for solving 2 point BVPs:

- Finite difference (FD) discretization → *sparse* linear system.
- Integral equation (IE) discretization → *dense* linear system.

How do the two methods compare?



Note: *For the IE, the condition number is small and constant!*

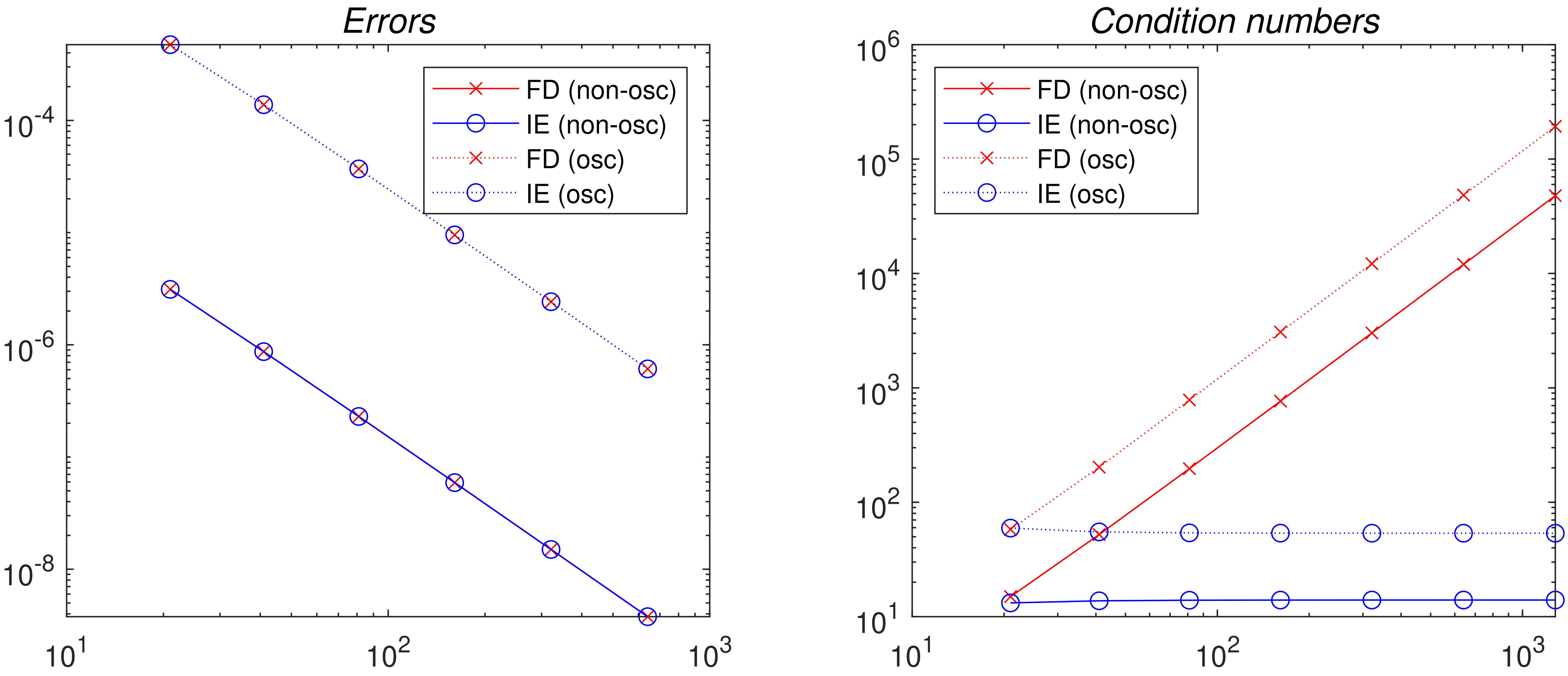
(Fun fact: The two methods are mathematically equivalent – errors are *identical!*)

Inversion of structured matrices: Semi-separable plus diagonal

Observe that we introduced two different approaches for solving 2 point BVPs:

- Finite difference (FD) discretization → *sparse* linear system.
- Integral equation (IE) discretization → *dense* linear system.

How do the two methods compare?



The conversion to an IE is an example of "analytic preconditioning".

You do the preconditioning mathematically, before you discretize.

Inversion of structured matrices: Semi-separable plus diagonal

Observe that we introduced two different approaches for solving 2 point BVPs:

- Finite difference (FD) discretization → *sparse* linear system.
- Integral equation (IE) discretization → *dense* linear system.

How do the two methods compare?

What about computational costs?

Inversion of structured matrices: Semi-separable plus diagonal

Observe that we introduced two different approaches for solving 2 point BVPs:

- Finite difference (FD) discretization → *sparse* linear system.
- Integral equation (IE) discretization → *dense* linear system.

How do the two methods compare?

What about computational costs?

Cost of computing an inverse is $O(n)$ in either case!

Inversion of structured matrices: Semi-separable plus diagonal

Observe that we introduced two different approaches for solving 2 point BVPs:

- Finite difference (FD) discretization → *sparse* linear system.
- Integral equation (IE) discretization → *dense* linear system.

How do the two methods compare?

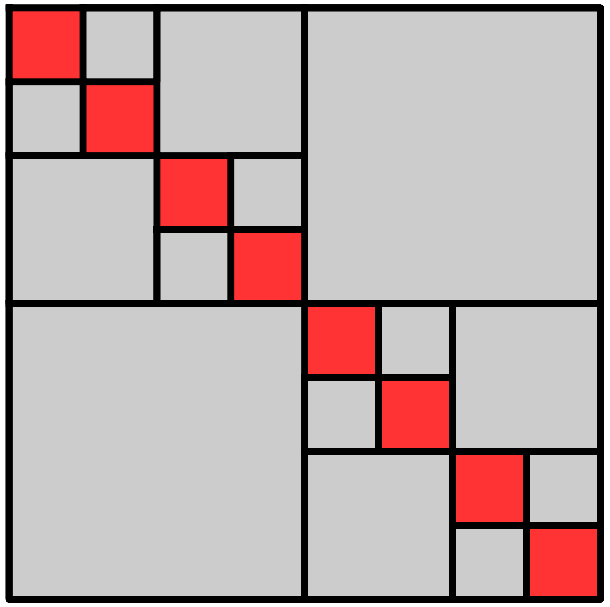
What about computational costs?

Cost of computing an inverse is $O(n)$ in either case!

We skip details for the “semi-separable plus diagonal” case, and instead go straight to a more general class: HODLR.

Inversion of structured matrices: HODLR

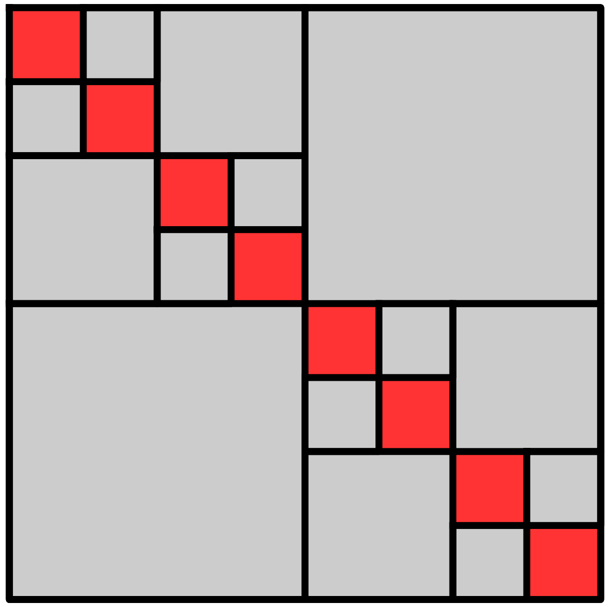
Now let us consider a slightly more complex structure:



All gray blocks have low rank.

Inversion of structured matrices: HODLR

Now let us consider a slightly more complex structure:



All gray blocks have low rank.

Let us start with a simple *recursive* inversion procedure.

The first step is to observe that if we tessellate $\mathbf{A}$ as follows

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

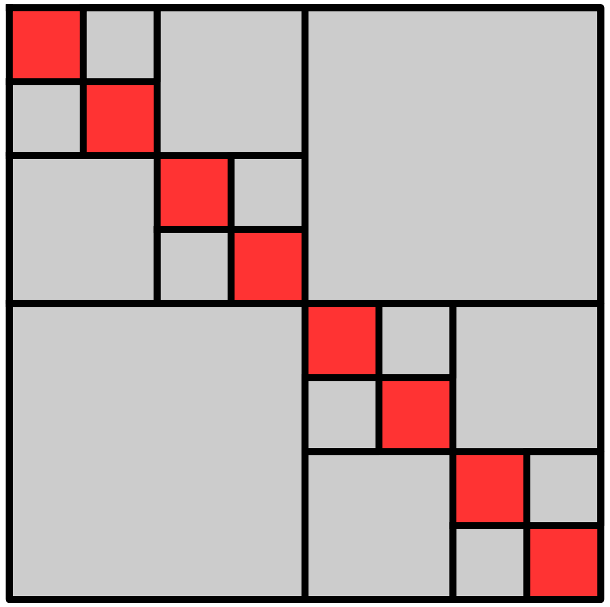
then

- $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ each have low numerical rank,
- $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ each are HODLR themselves.

Note: *"Semi-separable + diagonal" is a special case of HODLR.*

Inversion of structured matrices: HODLR

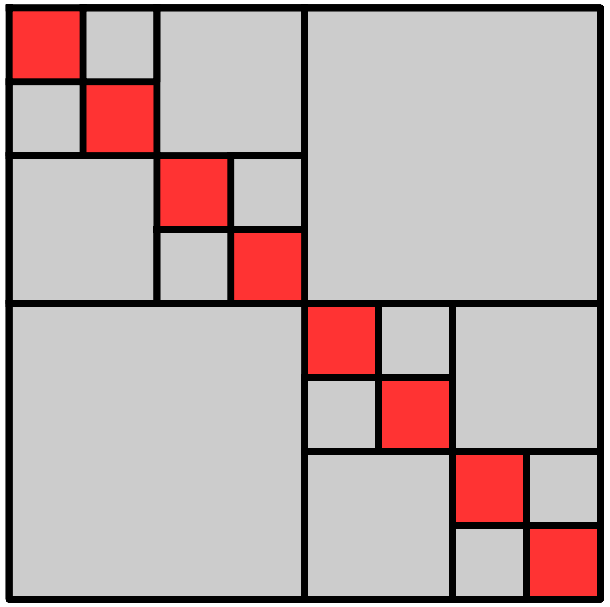
Now let us consider a slightly more complex structure:



All gray blocks have low rank.

Inversion of structured matrices: HODLR

Now let us consider a slightly more complex structure:



All gray blocks have low rank.

Let us start with a simple *recursive* inversion procedure.

The first step is to observe that if we tessellate **A** as follows

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

then

- $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ each have low numerical rank,
- $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ each are HODLR themselves.

Note: *"Semi-separable + diagonal" is a special case of HODLR.*

Inversion of structured matrices: HODLR

We seek to invert a matrix $\mathbf{A}$ as shown. Each block is of size $n \times n$, and $\mathbf{A}_{12}$ *and* $\mathbf{A}_{21}$ *have rank* $k < n$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

Inversion of structured matrices: HODLR

We seek to invert a matrix $\mathbf{A}$ as shown. Each block is of size $n \times n$, and $\mathbf{A}_{12}$ *and* $\mathbf{A}_{21}$ *have rank* $k < n$.

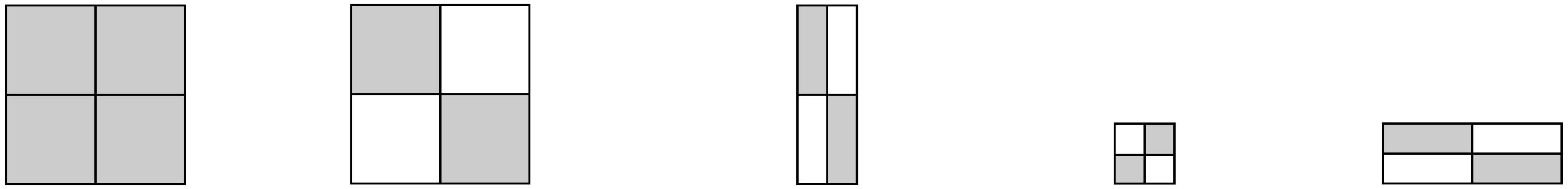
$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

We first form low-rank factorizations of $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ so that

$$\mathbf{A}_{12} = \mathbf{U}_1 \; \mathbf{B}_{12} \; \mathbf{V}_2^* \qquad \text{and} \qquad \mathbf{A}_{21} = \mathbf{U}_2 \; \mathbf{B}_{21} \; \mathbf{V}_1^*$$

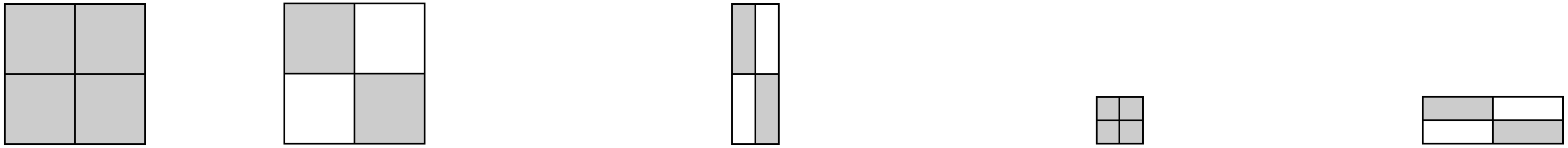
Then we can write $\mathbf{A}$ in the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^* \end{bmatrix}.$$



Applying the Woodbury formula, we get

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1} \end{bmatrix} + \begin{bmatrix} \mathbf{A}_{11}^{-1}\mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1}\mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{D}}_1 & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \hat{\mathbf{D}}_2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{V}_1^*\mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^*\mathbf{A}_{22}^{-1} \end{bmatrix},$$



where $\hat{\mathbf{D}}_1 = \left(\mathbf{V}_1^*\mathbf{A}_{11}^{-1}\mathbf{U}_1\right)^{-1}$ and $\hat{\mathbf{D}}_2 = \left(\mathbf{V}_2^*\mathbf{A}_{22}^{-1}\mathbf{U}_2\right)^{-1}$.

Inversion of structured matrices: HODLR

We seek to invert a matrix $\mathbf{A}$ as shown. Each block is of size $n \times n$, and $\mathbf{A}_{12}$ *and* $\mathbf{A}_{21}$ *have rank* $k < n$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

We first form low-rank factorizations of $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ so that

$$\mathbf{A}_{12} = \mathbf{U}_1\ \mathbf{B}_{12}\ \mathbf{V}_2^* \qquad \text{and} \qquad \mathbf{A}_{21} = \mathbf{U}_2\ \mathbf{B}_{21}\ \mathbf{V}_1^*$$

Then we can write $\mathbf{A}$ in the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^* \end{bmatrix}.$$

Applying the Woodbury formula, we get

$$\underset{2n \times 2n}{\mathbf{A}^{-1}} = \underset{2n \times 2n}{\begin{bmatrix} \mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1} \end{bmatrix}} + \underset{2n \times 2k}{\begin{bmatrix} \mathbf{A}_{11}^{-1}\mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1}\mathbf{U}_2 \end{bmatrix}} \underset{2k \times 2k}{\begin{bmatrix} \hat{\mathbf{D}}_1 & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \hat{\mathbf{D}}_2 \end{bmatrix}^{-1}} \underset{2k \times 2n}{\begin{bmatrix} \mathbf{V}_1^*\mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^*\mathbf{A}_{22}^{-1} \end{bmatrix}},$$

where $\hat{\mathbf{D}}_1 = \left(\mathbf{V}_1^*\mathbf{A}_{11}^{-1}\mathbf{U}_1\right)^{-1}$ and $\hat{\mathbf{D}}_2 = \left(\mathbf{V}_2^*\mathbf{A}_{22}^{-1}\mathbf{U}_2\right)^{-1}$.

Inversion of structured matrices: HODLR

We seek to invert a matrix $\mathbf{A}$ as shown. Each block is of size $n \times n$, and *$\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ have rank $k < n$.*

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

We first form low-rank factorizations of $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ so that

$$\mathbf{A}_{12} = \mathbf{U}_1\ \mathbf{B}_{12}\ \mathbf{V}_2^* \qquad \text{and} \qquad \mathbf{A}_{21} = \mathbf{U}_2\ \mathbf{B}_{21}\ \mathbf{V}_1^*$$

Then we can write $\mathbf{A}$ in the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^* \end{bmatrix}.$$

Applying the Woodbury formula, we get

$$\underbrace{\mathbf{A}^{-1}}_{2n \times 2n} = \underbrace{\begin{bmatrix} \mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1} \end{bmatrix}}_{2n \times 2n} + \underbrace{\begin{bmatrix} \mathbf{A}_{11}^{-1}\mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1}\mathbf{U}_2 \end{bmatrix}}_{2n \times 2k} \underbrace{\begin{bmatrix} \hat{\mathbf{D}}_1 & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \hat{\mathbf{D}}_2 \end{bmatrix}^{-1}}_{2k \times 2k} \underbrace{\begin{bmatrix} \mathbf{V}_1^*\mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^*\mathbf{A}_{22}^{-1} \end{bmatrix}}_{2k \times 2n},$$

where $\hat{\mathbf{D}}_1 = \left(\mathbf{V}_1^*\mathbf{A}_{11}^{-1}\mathbf{U}_1\right)^{-1}$ and $\hat{\mathbf{D}}_2 = \left(\mathbf{V}_2^*\mathbf{A}_{22}^{-1}\mathbf{U}_2\right)^{-1}$. So to get $\mathbf{A}^{-1}$, we need to:

- Compute $\mathbf{A}_{11}^{-1}$ and $\mathbf{A}_{22}^{-1}$. *Two inverses of half the size.*
- Form $\hat{\mathbf{D}}_1$ and $\hat{\mathbf{D}}_2$, and then invert $\begin{bmatrix} \hat{\mathbf{D}}_1 & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \hat{\mathbf{D}}_2 \end{bmatrix}$. *This is a small ($2k \times 2k$) matrix.*
- Form various matrix-matrix products involving at least one “thin” matrix.

Obvious recursion! *But: Rank growth.*

Inversion of structured matrices: HODLR

The recursive inversion formula for a HODLR matrix is conceptually simple.

But it is hard to code efficiently, and leads to growth of the numerical ranks.

Inversion of structured matrices: HODLR

The recursive inversion formula for a HODLR matrix is conceptually simple.

But it is hard to code efficiently, and leads to growth of the numerical ranks.

Luckily, there are better options, including non-recursive *exact* formulas.

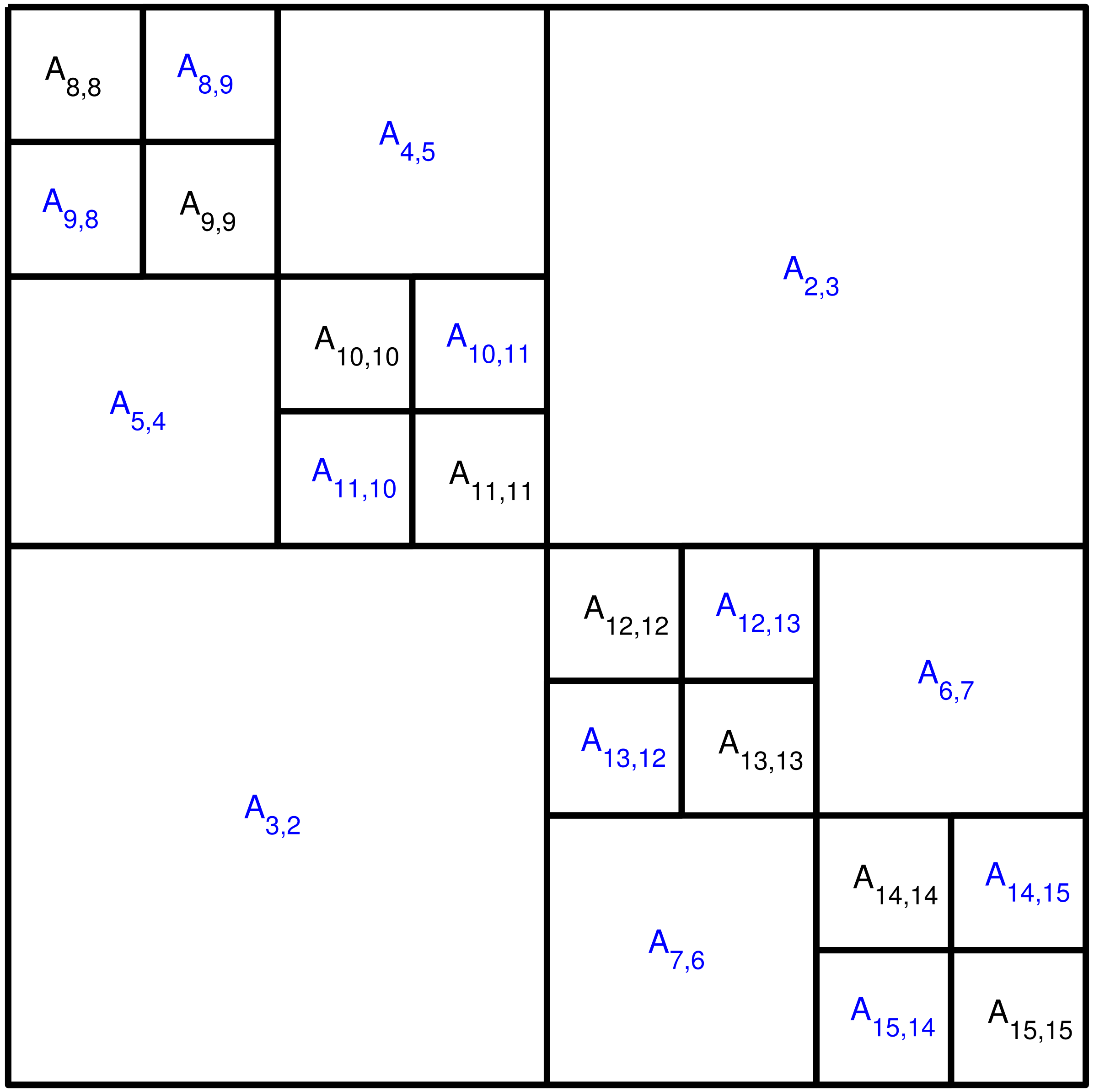
Inversion of structured matrices: HODLR

The recursive inversion formula for a HODLR matrix is conceptually simple.

But it is hard to code efficiently, and leads to growth of the numerical ranks.

Luckily, there are better options, including non-recursive *exact* formulas.

Let us consider a specific example:



For every sibling pair $\{\alpha, \beta\}$ set

$$\mathbf{A}_{\alpha,\beta} = \mathbf{A}(I_\alpha, I_\beta).$$

Fix a bound k on the rank, and a tolerance ε. We then require that each off-diagonal block (in blue in the figure) have ε-rank at most k. Define factors

$$\underset{n_\alpha \times n_\beta}{\mathbf{A}_{\alpha,\beta}} = \underset{n_\alpha \times k}{\mathbf{U}_\alpha}\ \underset{k \times n_\beta}{\mathbf{V}^*_\beta}.$$

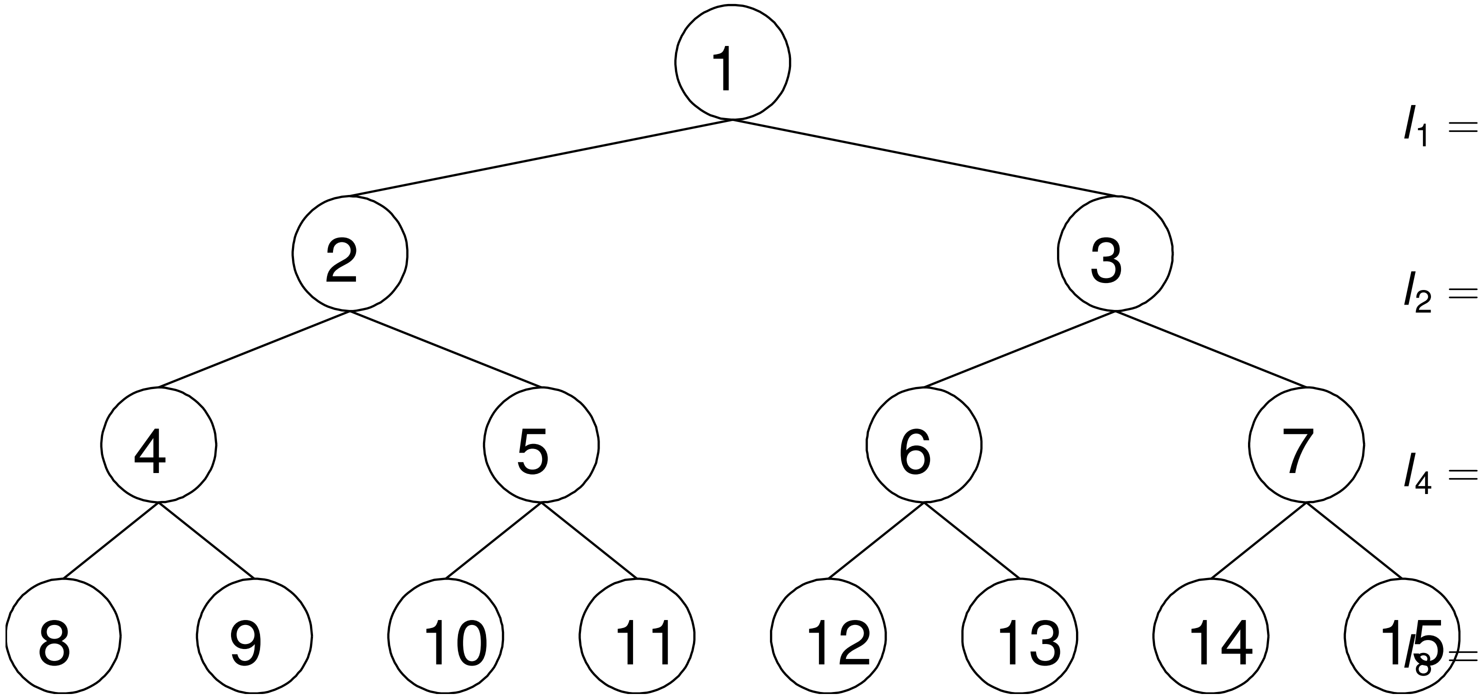
Inversion of structured matrices: HODLR

To more formally define the HODLR format, we need to introduce a *tree structure* on the index set $I = [1, 2, 3, \ldots, N]$.

Let $\mathbf{A}$ be an $N \times N$ matrix.

Suppose $\mathcal{T}$ is a binary tree on the index vector $I = [1, 2, 3, \ldots, N]$.

For a node τ in the tree, let I_τ denote the corresponding index vector.



Level 0 $I_1 = [1, 2, \ldots, 400]$

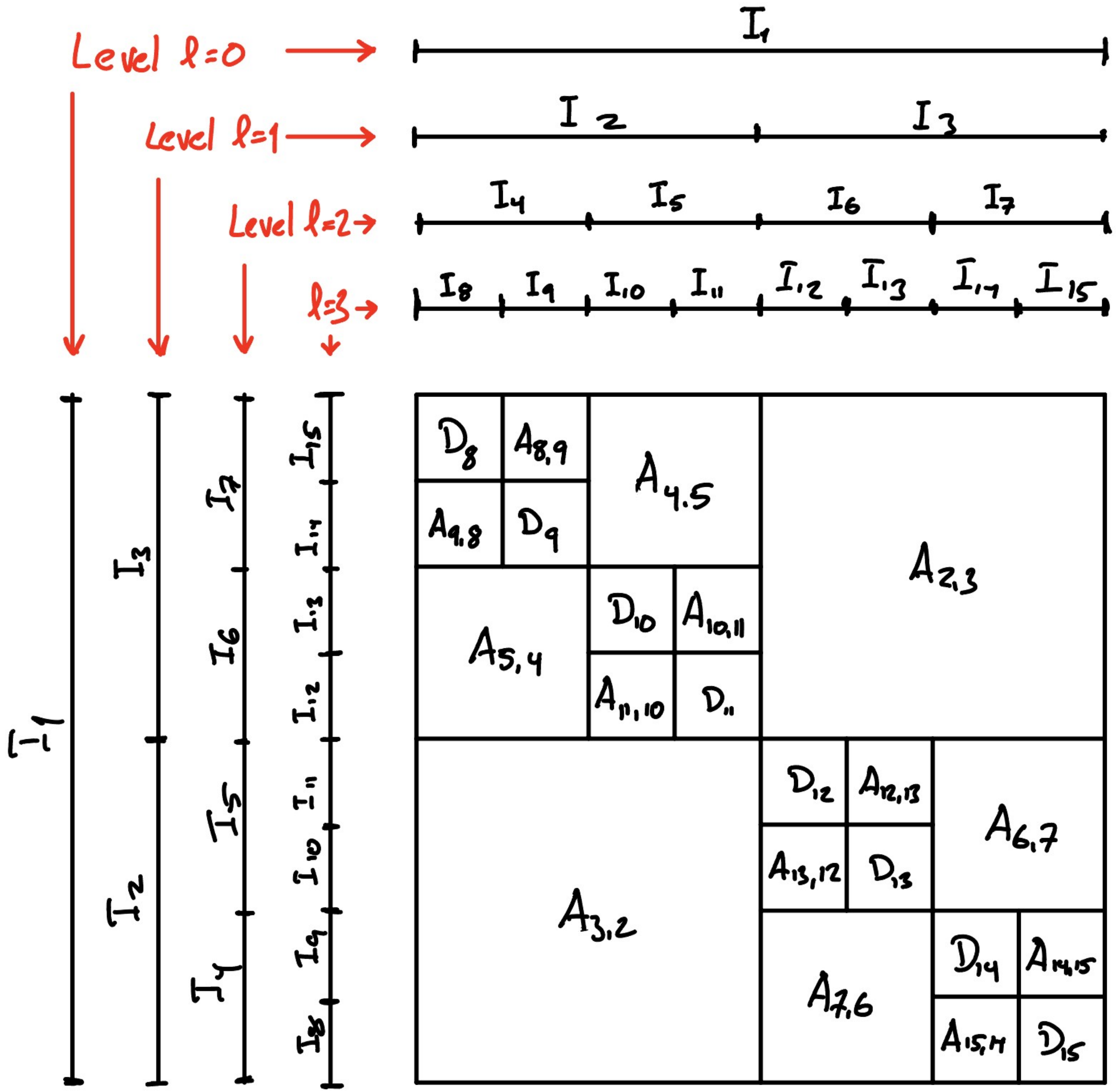
Level 1 $I_2 = [1, 2, \ldots, 200], I_3 = [201, 202, \ldots, 400]$

Level 2 $I_4 = [1, 2, \ldots, 100], I_5 = [101, 102, \ldots, 200], \ldots$

Level 3 $I_8 = [1, 2, \ldots, 50], I_9 = [51, 52, \ldots, 100], \ldots$

For nodes σ and τ on the same level, set $\mathbf{A}_{\sigma,\tau} = \mathbf{A}(I_\sigma, I_\tau)$.

Inversion of structured matrices: HODLR



Inversion of structured matrices: HODLR

For our 3-level model problem, we will build $\mathbf{A}^{-1}$ in the form

$$\mathbf{A}^{-1} = \mathbf{B}_0\,\mathbf{B}_1\,\mathbf{B}_2\,\mathbf{B}_3,$$

where each $\mathbf{B}_\ell$ is block diagonal, with diagonal blocks that are rank-k perturbations of the identity matrix. Consequently, each $\mathbf{B}_\ell$ can be applied to a vector in $O(Nk)$ flops.

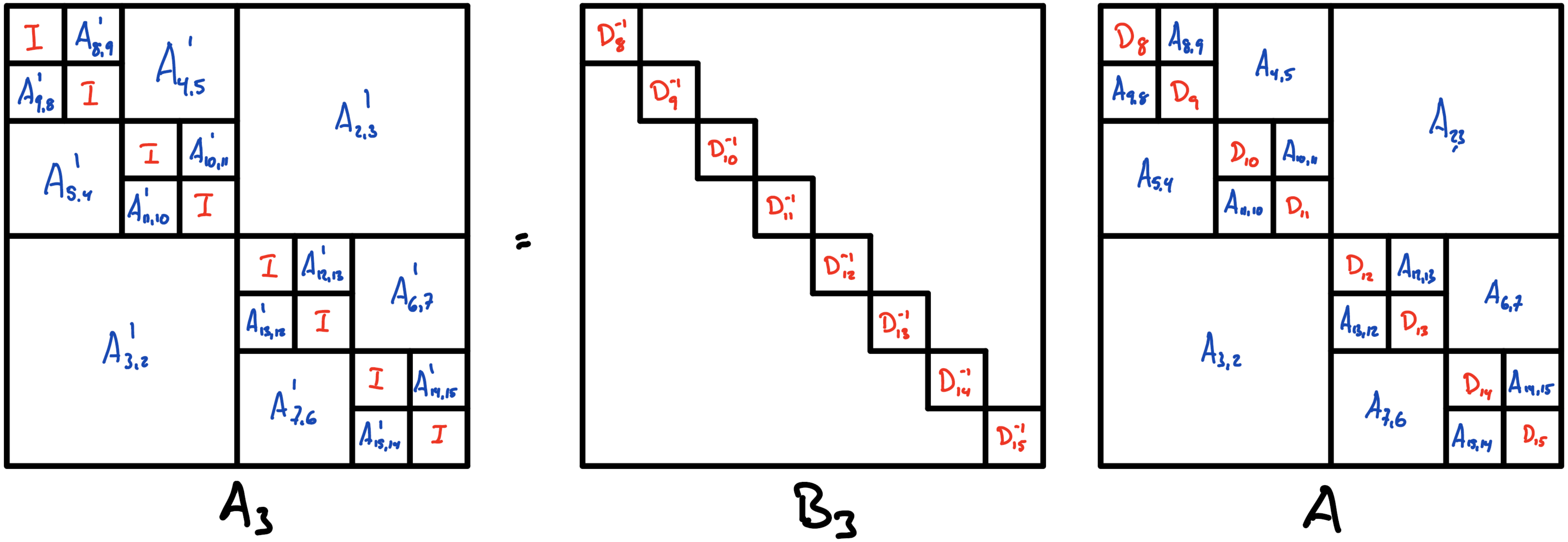
Inversion of structured matrices: HODLR

For our 3-level model problem, we will build $\mathbf{A}^{-1}$ in the form

$$\mathbf{A}^{-1} = \mathbf{B}_0\,\mathbf{B}_1\,\mathbf{B}_2\,\mathbf{B}_3,$$

where each $\mathbf{B}_\ell$ is block diagonal, with diagonal blocks that are rank-k perturbations of the identity matrix. Consequently, each $\mathbf{B}_\ell$ can be applied to a vector in $O(Nk)$ flops.

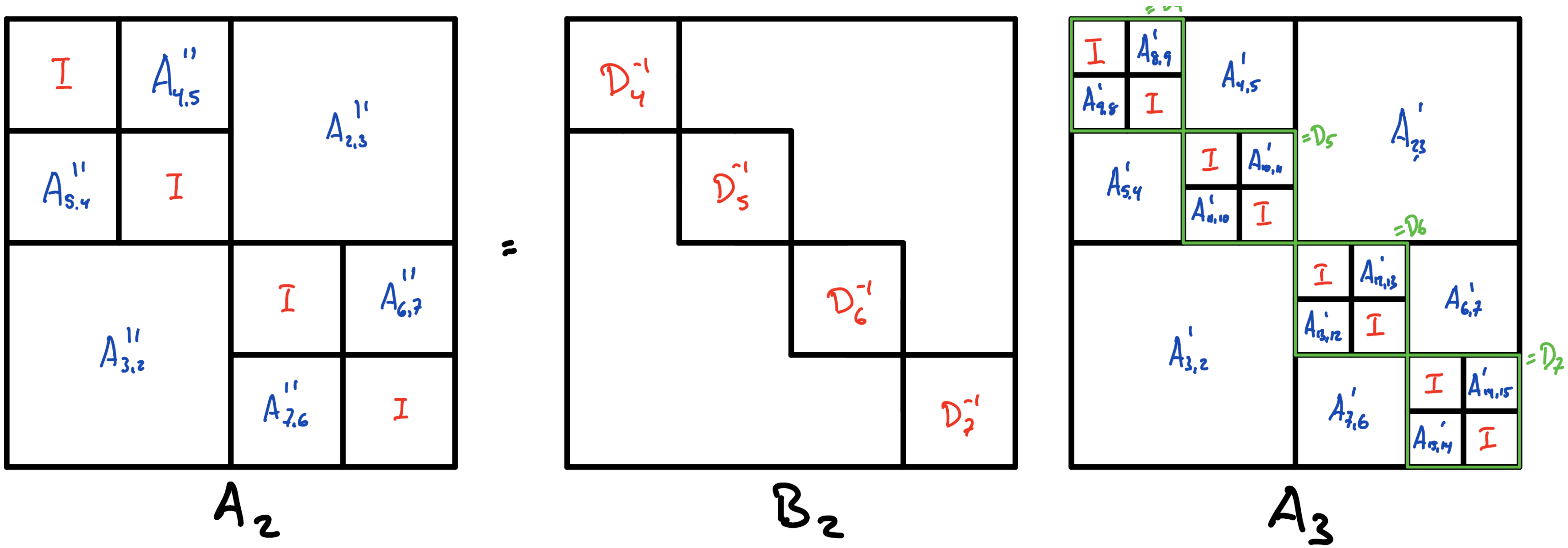
In the first step, we form a block diagonal matrix $\mathbf{B}_3$ whose diagonal blocks are the inverses of the diagonal blocks of $\mathbf{A}$. We then apply $\mathbf{B}_3$ to $\mathbf{A}$, to form a matrix $\mathbf{A}_3 = \mathbf{B}_3\mathbf{A}$ whose diagonal blocks are the identity matrix:



Observe that all the off-diagonal blocks in $\mathbf{A}_3$ still have rank at most k.

Inversion of structured matrices: HODLR

In the second step, let $\mathbf{D}_4$, $\mathbf{D}_5$, $\mathbf{D}_6$, $\mathbf{D}_7$ denote the diagonal blocks of $\mathbf{A}_3$, as marked in the figure below. Since each of these matrices are of the form “identity plus low rank”, they can inexpensively be inverted. We put the inverses $\mathbf{D}_4^{-1}$, $\mathbf{D}_5^{-1}$, $\mathbf{D}_6^{-1}$, $\mathbf{D}_7^{-1}$ into the diagonal blocks of $\mathbf{B}_2$ and apply it to $\mathbf{A}_3$ to obtain



Observe again that the off-diagonal blocks of $\mathbf{A}_2$ all have rank at most k.

Inversion of structured matrices: HODLR

The third step:

$$\underbrace{\begin{bmatrix} I & A'''_{2,3} \\ A'''_{3,2} & I \end{bmatrix}}_{A_1} = \underbrace{\begin{bmatrix} D_2^{-1} & \\ & D_3^{-1} \end{bmatrix}}_{B_1} \underbrace{\begin{bmatrix} \begin{bmatrix} I & A'_{4,5} \\ A'_{5,4} & I \end{bmatrix} & A'_{23} \\ A'_{3,2} & \begin{bmatrix} I & A'_{6,7} \\ A'_{7,6} & I \end{bmatrix} = D_3 \end{bmatrix}}_{A_2}$$

The fourth and final step:

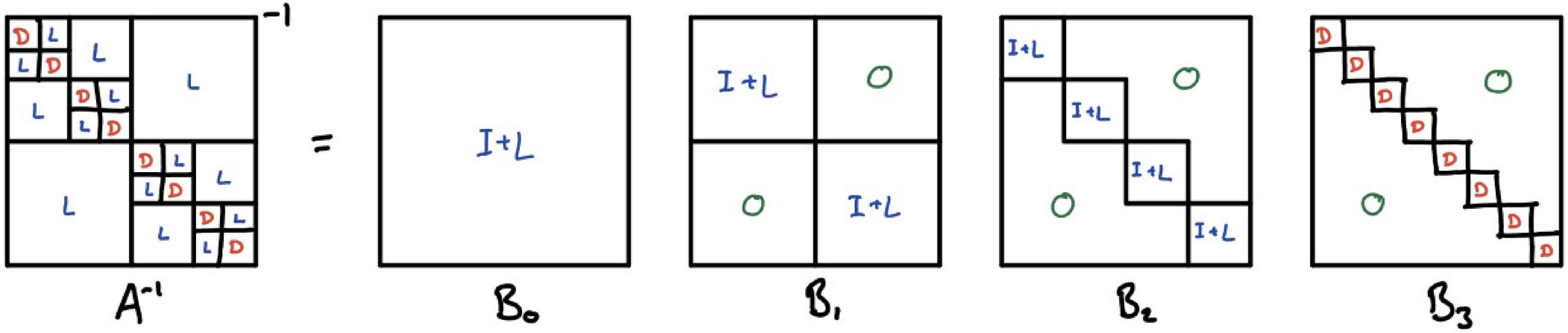
$$\underbrace{\begin{bmatrix} I \end{bmatrix}}_{I} = \underbrace{\begin{bmatrix} D_1^{-1} \end{bmatrix}}_{B_0} \underbrace{\begin{bmatrix} I & A'''_{23} \\ A'''_{3,2} & I \end{bmatrix}}_{A_1 = D_1}$$

Inversion of structured matrices: HODLR

Once the process completes, we have obtained the factorization

$$\mathbf{I} = \mathbf{B}_0\mathbf{A}_0 = \mathbf{B}_0\mathbf{B}_1\mathbf{A}_1 = \mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{A}_2 = \mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{A}.$$

In other words,



where

"L" means "low rank"

"D" means "dense"

All updates are ***multiplicative****.*

Overall complexity is $O(N(\log N)^2)$ for inversion.

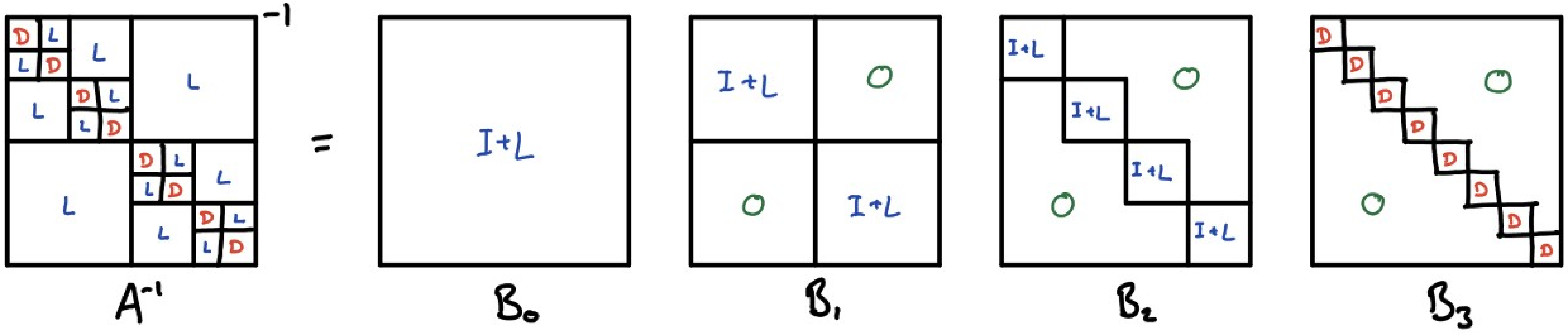
Very fast in practice. (Numerical examples shortly.)

Inversion of structured matrices: HODLR

Once the process completes, we have obtained the factorization

$$\mathbf{I} = \mathbf{B}_0\mathbf{A}_0 = \mathbf{B}_0\mathbf{B}_1\mathbf{A}_1 = \mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{A}_2 = \mathbf{B}_0\mathbf{B}_1\mathbf{B}_2\mathbf{B}_3\mathbf{A}.$$

In other words,



where

“L” means “low rank”

“D” means “dense”

All updates are ***multiplicative****.*

Overall complexity is $O(N(\log N)^2)$ for inversion.

Very fast in practice. (Numerical examples shortly.)

Next: Let us get rid of the log-factors!

Let us generalize from a matrix with 2×2 blocks to $p \times p$:

Suppose $\mathbf{A}$ is a “block-separable” matrix consisting of $p \times p$ blocks of size $n \times n$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14} \\ \mathbf{A}_{21} & \mathbf{D}_2 & \mathbf{A}_{23} & \mathbf{A}_{24} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{D}_3 & \mathbf{A}_{34} \\ \mathbf{A}_{41} & \mathbf{A}_{42} & \mathbf{A}_{43} & \mathbf{D}_4 \end{bmatrix}. \qquad \text{(Shown for } p = 4.\text{)}$$

Core assumption: Each off-diagonal block $\mathbf{A}_{ij}$ admits the factorization

$$\begin{array}{cccccc} \mathbf{A}_{ij} & = & \mathbf{U}_i & \tilde{\mathbf{A}}_{ij} & \mathbf{V}_j^* \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

where the rank k is significantly smaller than the block size n.

The critical part of the assumption is that all off-diagonal blocks in the i’th row use the same basis matrices $\mathbf{U}_i$ for their column spaces (and analogously all blocks in the j’th column use the same basis matrices $\mathbf{V}_j$ for their row spaces).

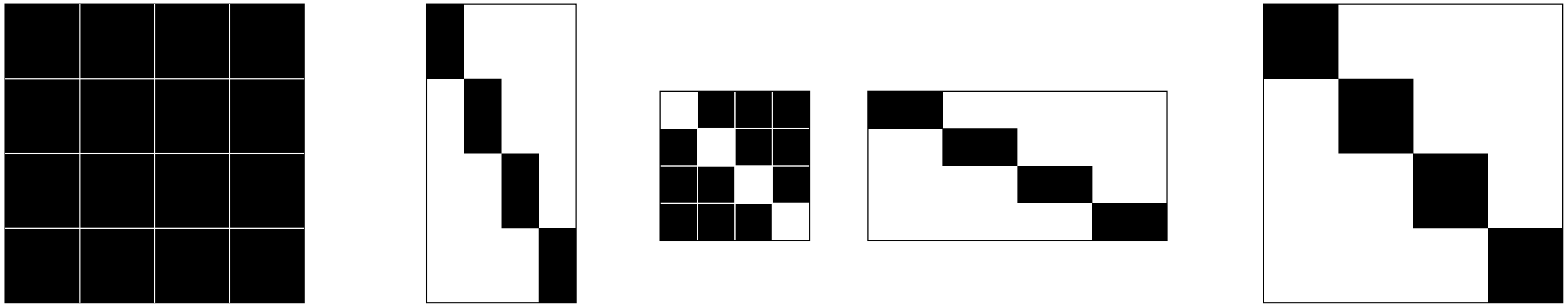
Recall $\mathbf{A} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{U}_1\,\tilde{\mathbf{A}}_{12}\,\mathbf{V}_2^* & \mathbf{U}_1\,\tilde{\mathbf{A}}_{13}\,\mathbf{V}_3^* & \mathbf{U}_1\,\tilde{\mathbf{A}}_{14}\,\mathbf{V}_4^* \\ \mathbf{U}_2\,\tilde{\mathbf{A}}_{21}\,\mathbf{V}_1^* & \mathbf{D}_2 & \mathbf{U}_2\,\tilde{\mathbf{A}}_{23}\,\mathbf{V}_3^* & \mathbf{U}_2\,\tilde{\mathbf{A}}_{24}\,\mathbf{V}_4^* \\ \mathbf{U}_3\,\tilde{\mathbf{A}}_{31}\,\mathbf{V}_1^* & \mathbf{U}_3\,\tilde{\mathbf{A}}_{32}\,\mathbf{V}_2^* & \mathbf{D}_3 & \mathbf{U}_3\,\tilde{\mathbf{A}}_{34}\,\mathbf{V}_4^* \\ \mathbf{U}_4\,\tilde{\mathbf{A}}_{41}\,\mathbf{V}_1^* & \mathbf{U}_4\,\tilde{\mathbf{A}}_{42}\,\mathbf{V}_2^* & \mathbf{U}_4\,\tilde{\mathbf{A}}_{43}\,\mathbf{V}_3^* & \mathbf{D}_4 \end{bmatrix}.$

Then $\mathbf{A}$ admits the factorization:

$$\mathbf{A} = \underbrace{\begin{bmatrix} \mathbf{U}_1 & & & \\ & \mathbf{U}_2 & & \\ & & \mathbf{U}_3 & \\ & & & \mathbf{U}_4 \end{bmatrix}}_{=\mathbf{U}} \underbrace{\begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_{12} & \tilde{\mathbf{A}}_{13} & \tilde{\mathbf{A}}_{14} \\ \tilde{\mathbf{A}}_{21} & \mathbf{0} & \tilde{\mathbf{A}}_{23} & \tilde{\mathbf{A}}_{24} \\ \tilde{\mathbf{A}}_{31} & \tilde{\mathbf{A}}_{32} & \mathbf{0} & \tilde{\mathbf{A}}_{34} \\ \tilde{\mathbf{A}}_{41} & \tilde{\mathbf{A}}_{42} & \tilde{\mathbf{A}}_{43} & \mathbf{0} \end{bmatrix}}_{=\tilde{\mathbf{A}}} \underbrace{\begin{bmatrix} \mathbf{V}_1^* & & & \\ & \mathbf{V}_2^* & & \\ & & \mathbf{V}_3^* & \\ & & & \mathbf{V}_4^* \end{bmatrix}}_{=\mathbf{V}^*} + \underbrace{\begin{bmatrix} \mathbf{D}_1 & & & \\ & \mathbf{D}_2 & & \\ & & \mathbf{D}_3 & \\ & & & \mathbf{D}_4 \end{bmatrix}}_{=\mathbf{D}}$$

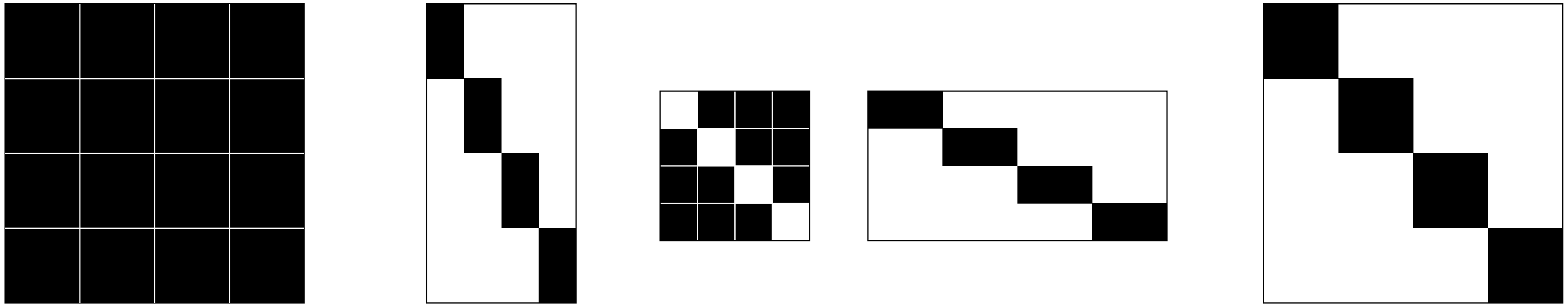
or

$$\begin{array}{ccccccc} \mathbf{A} & = & \mathbf{U} & \tilde{\mathbf{A}} & \mathbf{V}^* & + & \mathbf{D}, \\ pn \times pn & & pn \times pk & pk \times pk & pk \times pn & & pn \times pn \end{array}$$



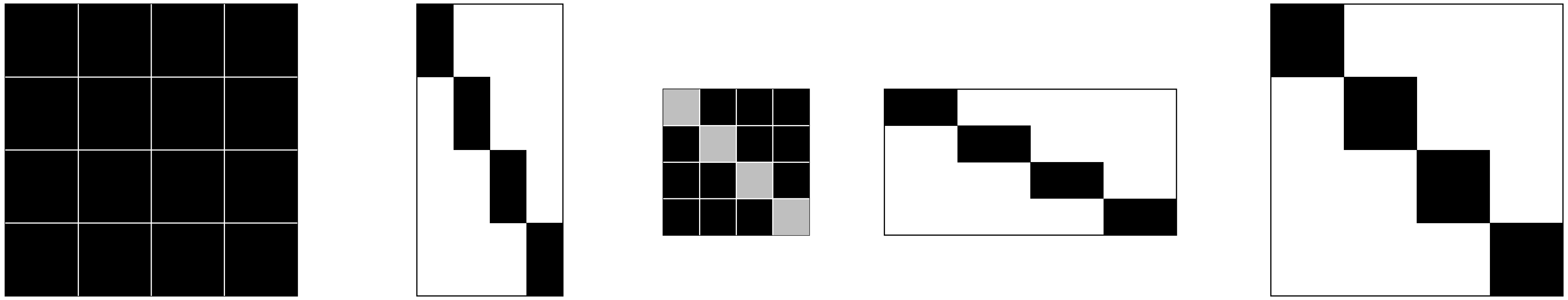
Lemma: [Variation of Woodbury] If an $N \times N$ matrix $\mathbf{A}$ admits the factorization

$$\begin{array}{ccccccc} \mathbf{A} & = & \mathbf{U} & \tilde{\mathbf{A}} & \mathbf{V}^* & + & \mathbf{D}, \\ pn \times pn & & pn \times pk & pk \times pk & pk \times pn & & pn \times pn \end{array}$$



then

$$\begin{array}{ccccccc} \mathbf{A}^{-1} & = & \mathbf{E} & (\tilde{\mathbf{A}} + \hat{\mathbf{D}})^{-1} & \mathbf{F}^* & + & \mathbf{G}, \\ pn \times pn & & pn \times pk & pk \times pk & pk \times pn & & pn \times pn \end{array}$$



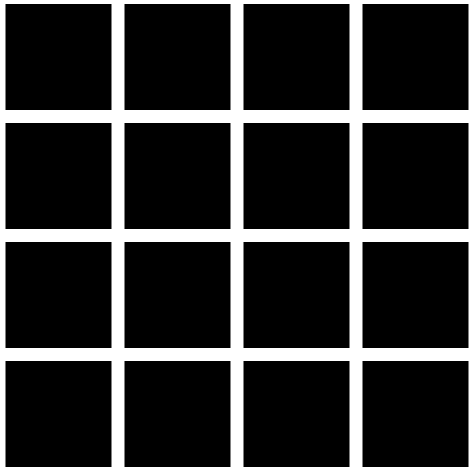
where (provided all intermediate matrices are invertible)

$$\hat{\mathbf{D}} = \left(\mathbf{V}^*\,\mathbf{D}^{-1}\,\mathbf{U}\right)^{-1}, \quad \mathbf{E} = \mathbf{D}^{-1}\,\mathbf{U}\,\hat{\mathbf{D}}, \quad \mathbf{F} = (\hat{\mathbf{D}}\,\mathbf{V}^*\,\mathbf{D}^{-1})^*, \quad \mathbf{G} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\,\mathbf{U}\,\hat{\mathbf{D}}\,\mathbf{V}^*\,\mathbf{D}^{-1}.$$

Note: All matrices set in blue are block diagonal.

Before compression, we have a $pn \times pn$ linear system

$$\sum_{j=1}^{p} \mathbf{A}_{ij}\mathbf{q}_j = \mathbf{f}_i, \quad i = 1, 2, \dots, p.$$



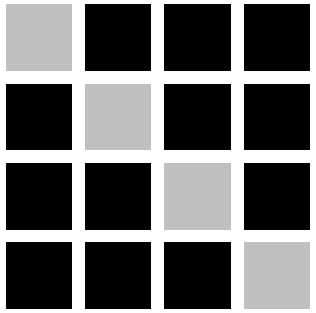
The original $4n \times 4n$ matrix.

After compression, we have a $pk \times pk$ linear system

$$\mathbf{D}_{ii}\tilde{\mathbf{q}}_i + \sum_{i \neq j} \tilde{\mathbf{A}}_{ij}\tilde{\mathbf{q}}_j = \tilde{\mathbf{f}}_i, \quad i = 1, 2, \dots, p.$$

Recall that k is the ε-rank of $\mathbf{A}_{i,j}$ for $i \neq j$.

The point is that $k < n$.



The reduced $4k \times 4k$ matrix.

The compression algorithm needs to execute the following steps:

- Compute $\mathbf{U}_i$, $\mathbf{V}_i$, $\tilde{\mathbf{A}}_{ij}$ so that $\mathbf{A}_{ij} = \mathbf{U}_i\,\tilde{\mathbf{A}}_{ij}\,\mathbf{V}_j^*$.
- Compute the new diagonal matrices $\hat{\mathbf{D}}_{ii} = \big(\mathbf{V}_i^*\,\mathbf{A}_{ii}^{-1}\,\mathbf{U}_i\big)^{-1}$.
- Compute the new loads $\tilde{\mathbf{q}}_i = \hat{\mathbf{D}}_{ii}\,\mathbf{V}_i^*\,\mathbf{A}_{ii}^{-1}\,\mathbf{q}_i$.

For the algorithm to be efficient, it has to be able to carry out these steps *locally*.

To achieve this, we use interpolative decompositions, then $\tilde{\mathbf{A}}_{i,j} = \mathbf{A}(\tilde{I}_i, \tilde{I}_j)$.

The Interpolative Decomposition (ID):

Let $\mathbf{B}$ be an $m \times n$ matrix of (precise) rank k. Then $\mathbf{B}$ admits a factorization

$$\begin{array}{cccc} \mathbf{B} & = \quad \mathbf{U} & \mathbf{B}^{(\mathrm{skel})} & \mathbf{V}^*, \\ m \times n & m \times k & k \times k & k \times n \end{array}$$

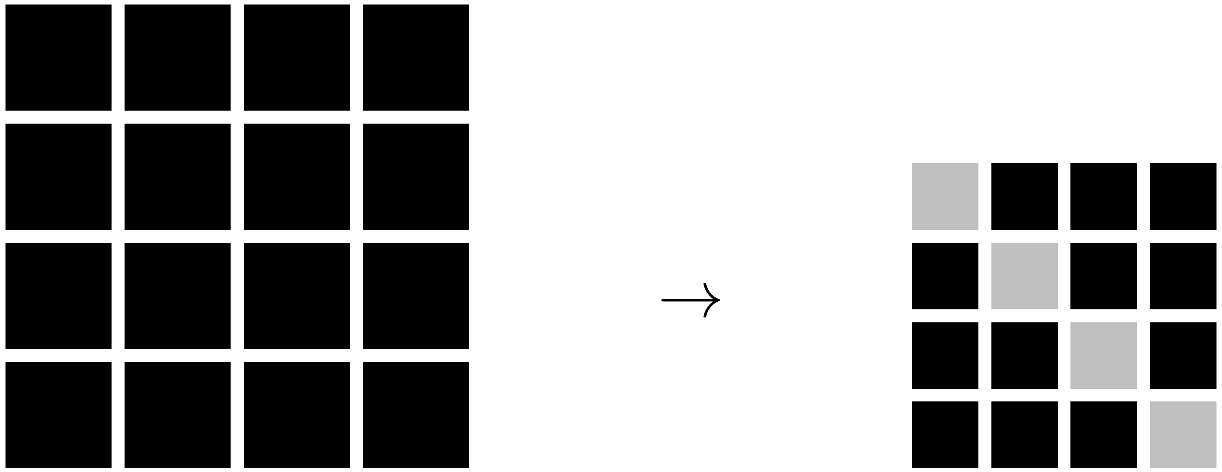
where

1. $\mathbf{B}^{(\mathrm{skel})} = \mathbf{B}(I_{\mathrm{row}}, I_{\mathrm{col}})$ is a *submatrix* of $\mathbf{B}$
2. $\mathbf{U}$ and $\mathbf{V}$ both contain a $k \times k$ identity matrix.
3. No entry of $\mathbf{U}$ or $\mathbf{V}$ has magnitude greater than 1 (so $\mathbf{U}$ and $\mathbf{V}$ are well-conditioned).

How do you construct an ID in practice?

- Computing an ID that satisfies (3) is (in general) very hard.
- If we relax condition (3) slightly, and require only that, say, $\max_{ij} |\mathbf{V}(i,j)| \leq 1.1$, then it can be done efficiently [1996, Gu & Eisenstat].
- In practice, simply performing Gram-Schmidt on the columns works great.
- When $\mathbf{B}$ has *approximate* rank k, the accuracy is excellent as long as the singular values of $\mathbf{B}$ decay rapidly. (They do in our applications.)

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



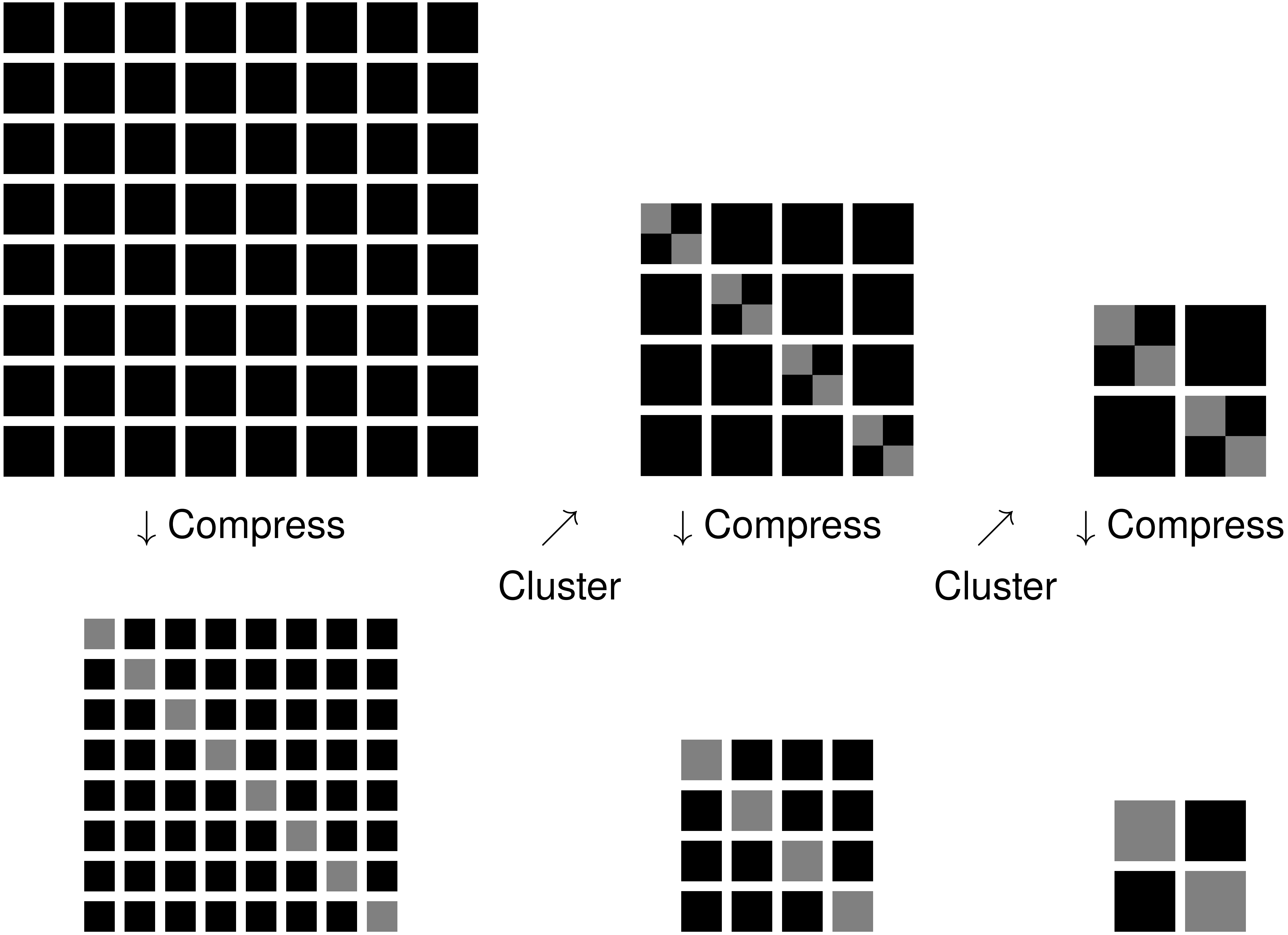
Important: *The black blocks are **submatrices**. No need to compute them!*

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!

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



↓ Compress

↗ Cluster

↓ Compress

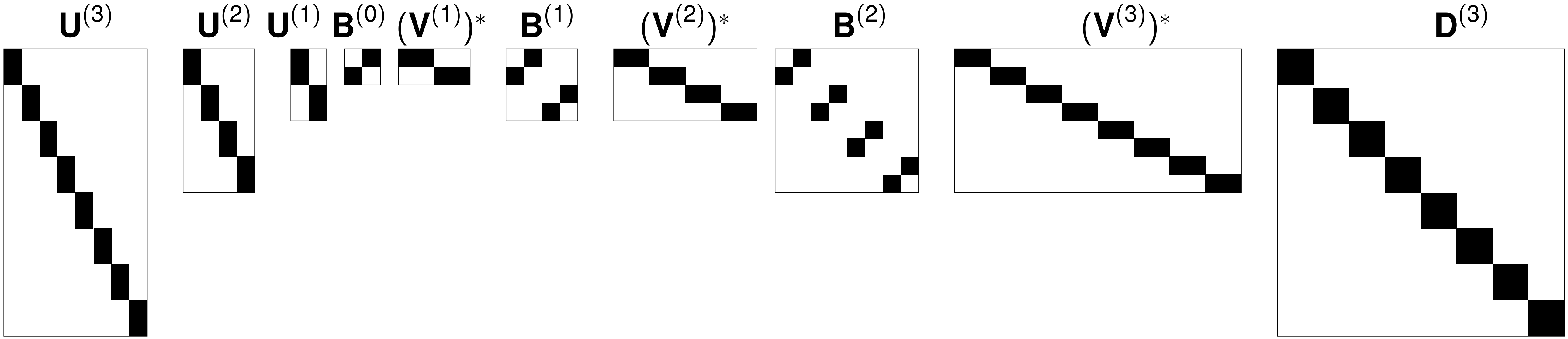
↗ Cluster

↓ Compress

Formally, one can view this as a telescoping factorization of **A**:

$$\mathbf{A} = \mathbf{U}^{(3)}\big(\mathbf{U}^{(2)}\big(\mathbf{U}^{(1)}\,\mathbf{B}^{(0)}\,(\mathbf{V}^{(1)})^{*} + \mathbf{B}^{(1)}\big)(\mathbf{V}^{(2)})^{*} + \mathbf{B}^{(2)}\big)(\mathbf{V}^{(3)})^{*} + \mathbf{D}^{(3)}.$$

Expressed pictorially, the factorization takes the form



The *inverse of A* then takes the form

$$\mathbf{A}^{-1} = \mathbf{E}^{(3)}\big(\mathbf{E}^{(2)}\big(\mathbf{E}^{(1)}\,\hat{\mathbf{D}}^{(0)}\,(\mathbf{F}^{(1)})^{*} + \hat{\mathbf{D}}^{(1)}\big)(\mathbf{F}^{(2)})^{*} + \hat{\mathbf{D}}^{(2)}\big)(\mathbf{V}^{(3)})^{*} + \hat{\mathbf{D}}^{(3)}.$$

All matrices are block diagonal except $\hat{\mathbf{D}}^{(0)}$*, which is small.*

Important: The inversion is *exact* up to floating point arithmetic!

(When we move to *strong* admissibility, no such formulas exist.)

Note:

The formulas we use in this talk were chosen because they are particularly clean and simple.

There is a slight reformulation of the scheme that is both more computationally efficient, and more numerically stable. The key is to never form the diagonal blocks explicitly. Recall:

$$\hat{\mathbf{D}}_\tau = \left(\mathbf{V}^*\mathbf{A}_\tau^{-1}\mathbf{U}_\tau\right)^{-1}$$

You can instead assemble just the “scattering matrix”

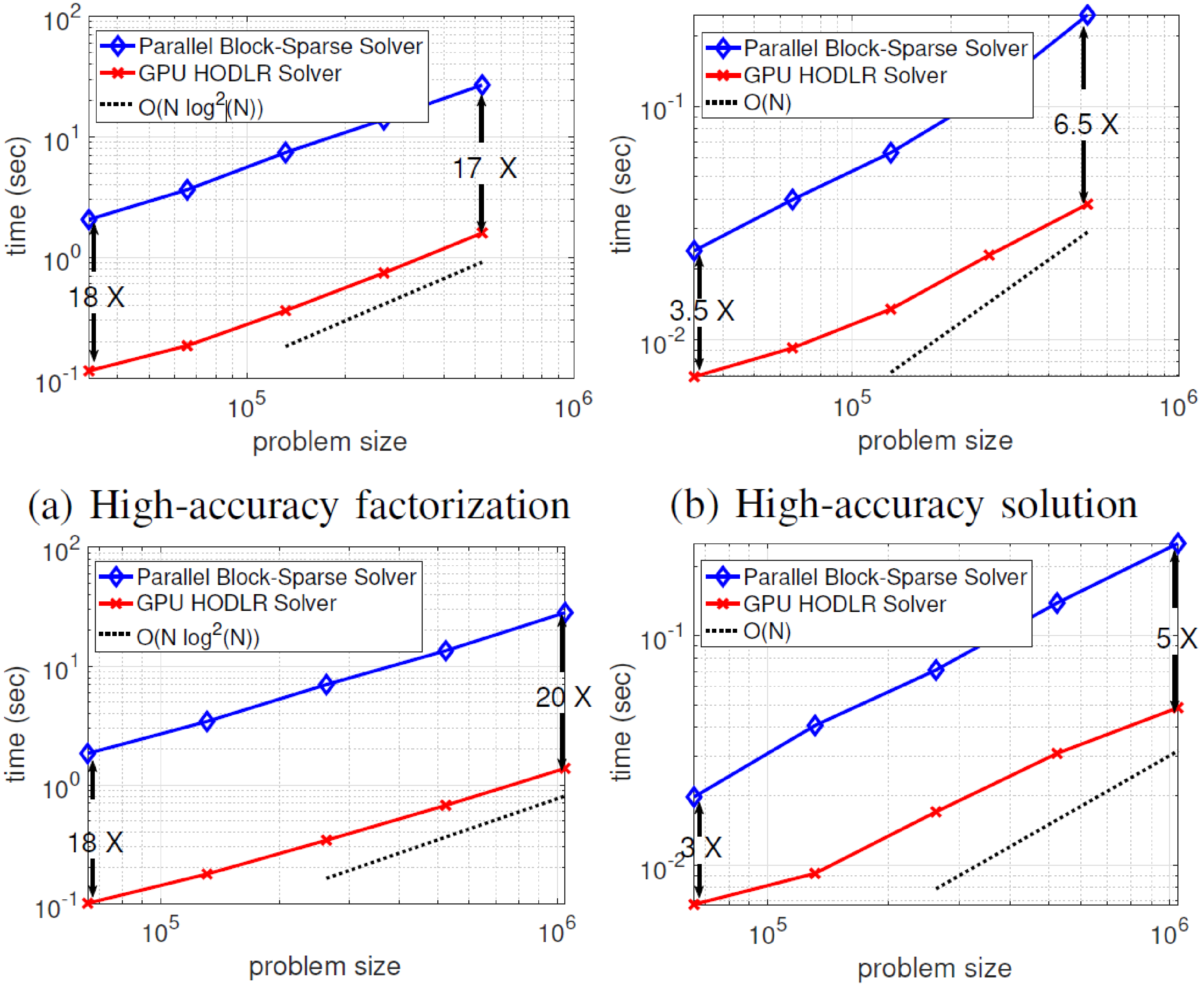
$$\mathbf{S}_\tau = \mathbf{V}^*\mathbf{A}_\tau^{-1}\mathbf{U}_\tau.$$

Chapter 18, “Fast Direct Solvers for Elliptic PDEs”, P.G. Martinsson, 2019

Inversion of an HBS matrix

```
loop over all levels, finer to coarser, $\ell = L, L-1, \dots, 1$
  loop over all boxes $\tau$ on level $\ell$,
    if $\tau$ is a leaf node
      $\mathbf{X} = \mathbf{D}_\tau$
    else
      Let $\sigma_1$ and $\sigma_2$ denote the children of $\tau$.
      $\mathbf{X} = \begin{bmatrix} \mathbf{D}_{\sigma_1} & \mathbf{B}_{\sigma_1,\sigma_2} \\ \mathbf{B}_{\sigma_2,\sigma_1} & \mathbf{D}_{\sigma_2} \end{bmatrix}$
    end if
    $\mathbf{D}_\tau = \left(\mathbf{V}_\tau^* \, \mathbf{X}^{-1} \, \mathbf{U}_\tau\right)^{-1}$.
    $\mathbf{E}_\tau = \mathbf{X}^{-1} \, \mathbf{U}_\tau \, \mathbf{D}_\tau$.
    $\mathbf{F}_\tau^* = \mathbf{D}_\tau \, \mathbf{V}_\tau^* \, \mathbf{X}^{-1}$.
    $\mathbf{G}_\tau = \mathbf{X}^{-1} - \mathbf{X}^{-1} \, \mathbf{U}_\tau \, \mathbf{D}_\tau \, \mathbf{V}_\tau^* \, \mathbf{X}^{-1}$.
  end loop
end loop
$\mathbf{G}_1 = \begin{bmatrix} \mathbf{D}_2 & \mathbf{B}_{2,3} \\ \mathbf{B}_{3,2} & \mathbf{D}_3 \end{bmatrix}^{-1}$.
```

Example: 2D Boundary integral equation for Helmholtz equation (HODLR)



(a) High-accuracy factorization

(b) High-accuracy solution

(c) Low-accuracy factorization

(d) Low-accuracy solution

The domain is about 20λ so ranks are now higher at the top levels.

Example: Lippmann-Schwinger

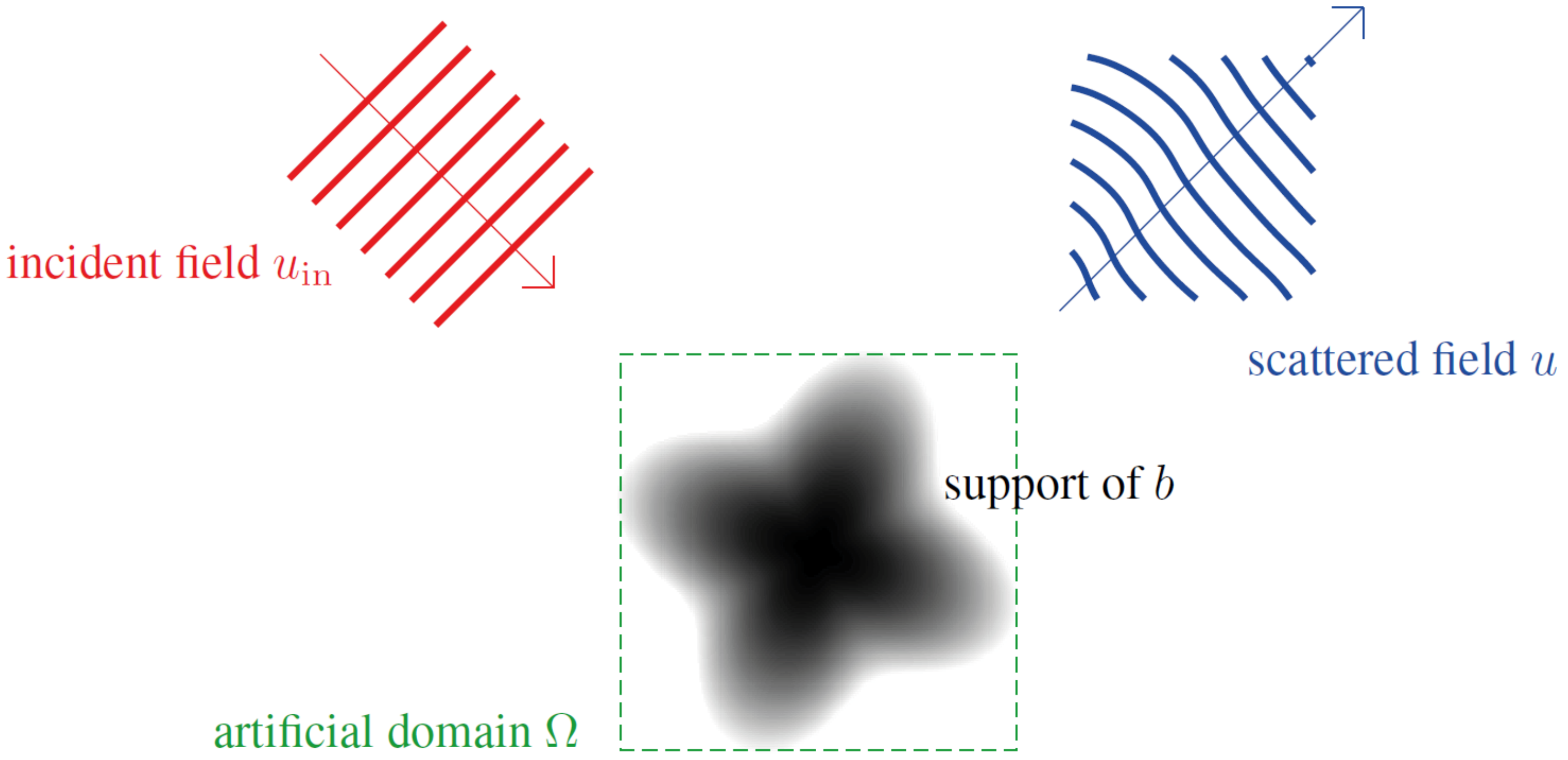
With Abi Gopal

Consider the free space acoustic scattering problem

$$(9)\qquad \begin{cases} -\Delta u(\boldsymbol{x}) - \kappa^2 (1 - b(\boldsymbol{x}))\, u(\boldsymbol{x}) = -\,\kappa^2\, b(\boldsymbol{x})\, v(\boldsymbol{x}), & \boldsymbol{x} \in \mathbb{R}^2 \\ \lim\limits_{|\boldsymbol{x}|\to\infty} \sqrt{|\boldsymbol{x}|}\big(\partial_{|\boldsymbol{x}|} u(\boldsymbol{x}) - i\kappa\, u(\boldsymbol{x})\big) = 0, & \end{cases}$$

where

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



In the figure, $v = u_{\text{in}}$.

Example: Lippmann-Schwinger

With Abi Gopal

Consider the free space acoustic scattering problem

$$(9) \qquad \begin{cases} -\Delta u(\boldsymbol{x}) - \kappa^2\,(1 - b(\boldsymbol{x}))\,u(\boldsymbol{x}) = \,-\,\kappa^2\,b(\boldsymbol{x})\,v(\boldsymbol{x}), & \boldsymbol{x} \in \mathbb{R}^2 \\ \lim\limits_{|\boldsymbol{x}|\to\infty} \sqrt{|\boldsymbol{x}|}\big(\partial_{|\boldsymbol{x}|} u(\boldsymbol{x}) - i\kappa\,u(\boldsymbol{x})\big) = 0, & \end{cases}$$

where

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

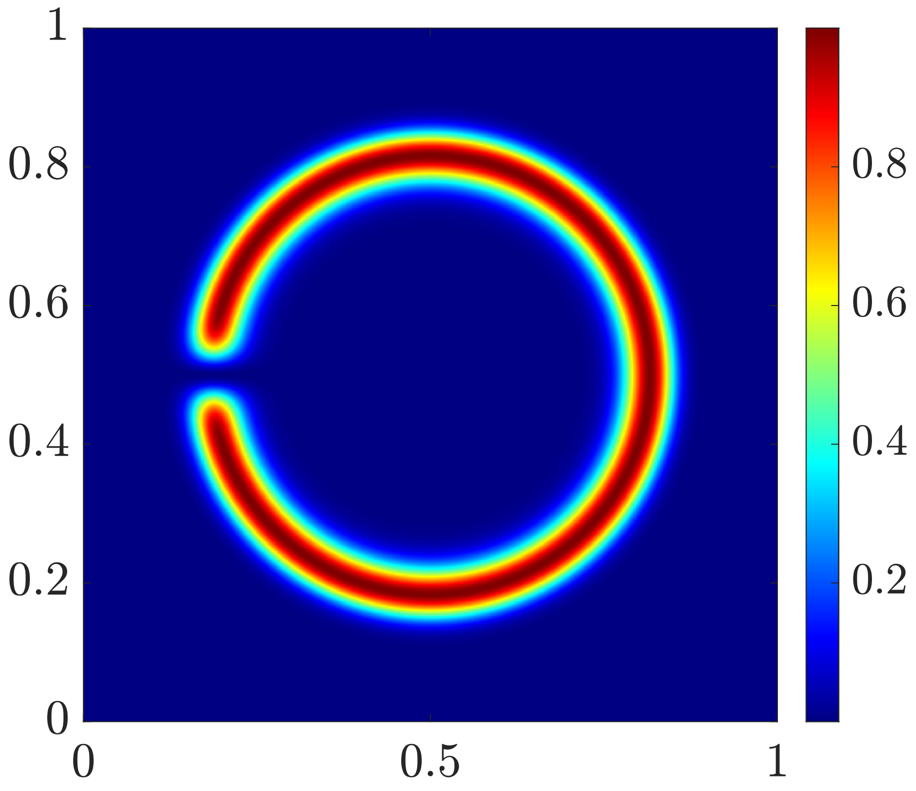
Let us now use an integral equation formulation. It is well known that (9) has an alternative formulation in the *Lippmann-Schwinger integral equation*

$$(10) \qquad \sigma(\boldsymbol{x}) + \kappa^2 b(\boldsymbol{x}) \int_\Omega \phi_\kappa(\boldsymbol{x} - \boldsymbol{y})\sigma(\boldsymbol{y})\,d\boldsymbol{y} = -\kappa^2 b(\boldsymbol{x}) v(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega,$$

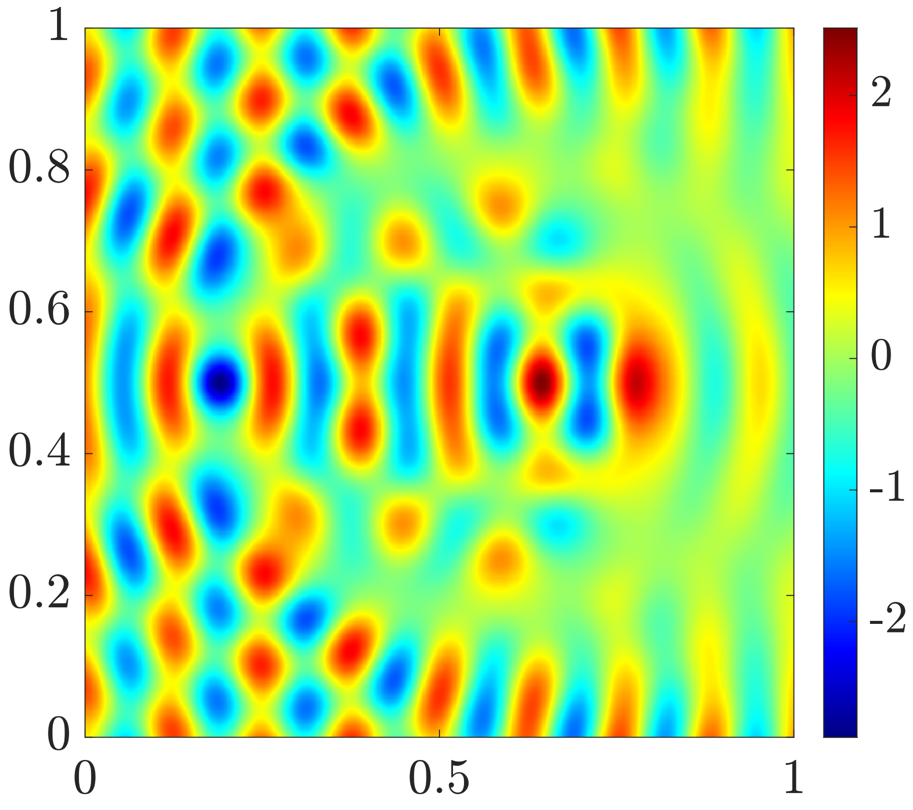
where ϕ_κ is the free space fundamental solution of the Helmholtz equation.

We discretize (7) using the trapezoidal rule on a uniform grid, with Duan-Rokhlin quadrature corrections of order 10.

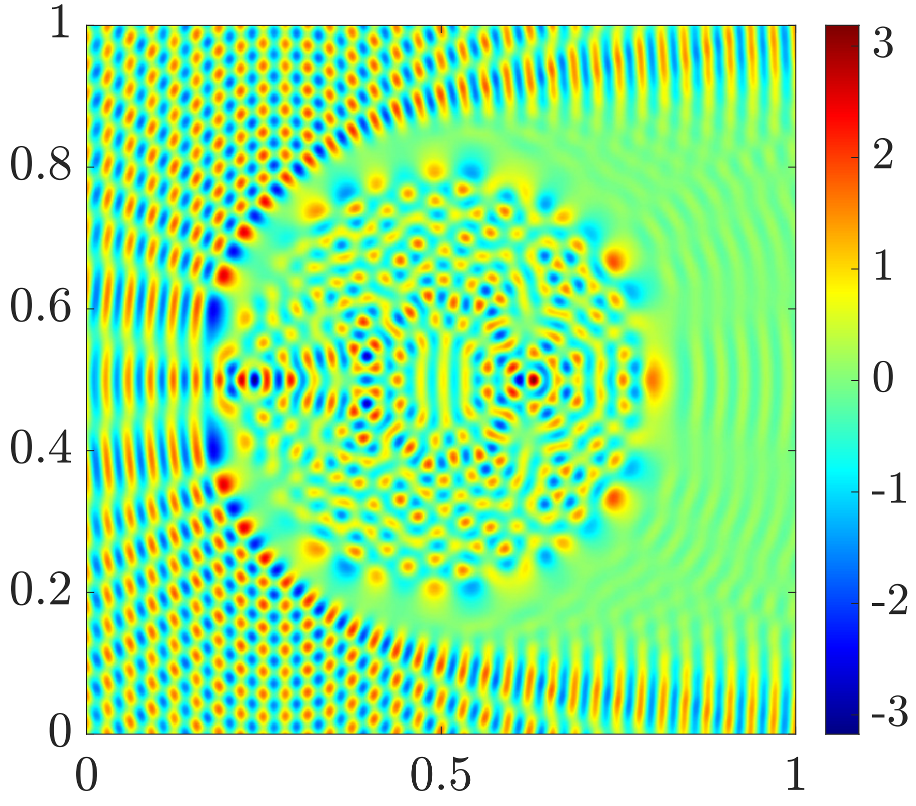
Example: Lippmann-Schwinger



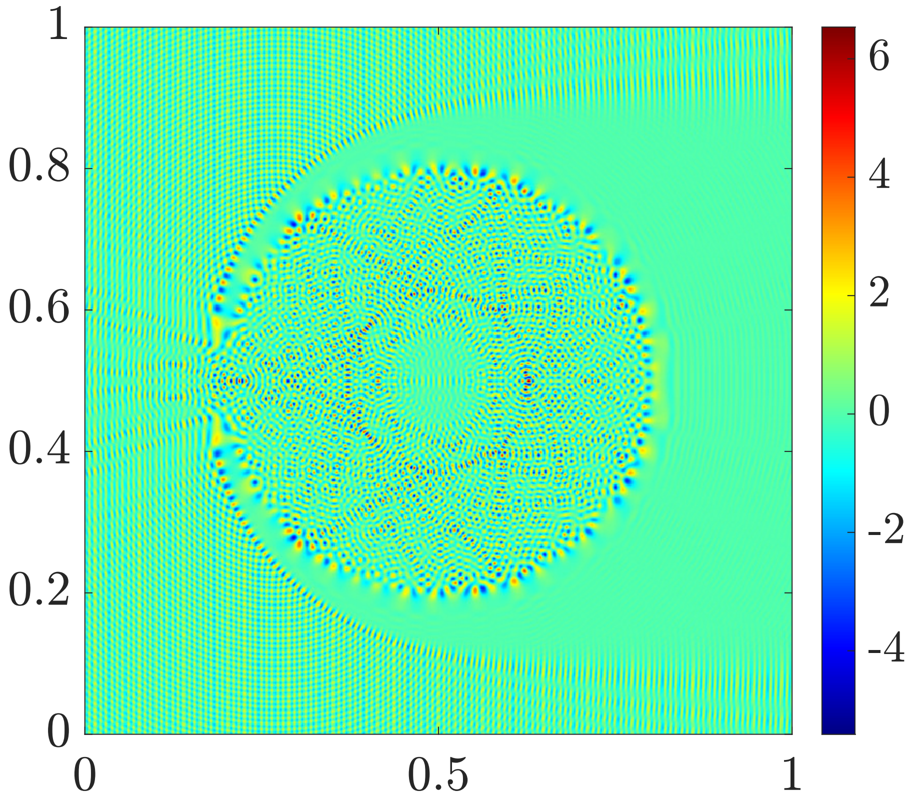
The scattering potential



$\kappa = 50$



$\kappa = 201$



$\kappa = 804$

Example: Lippmann-Schwinger

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).

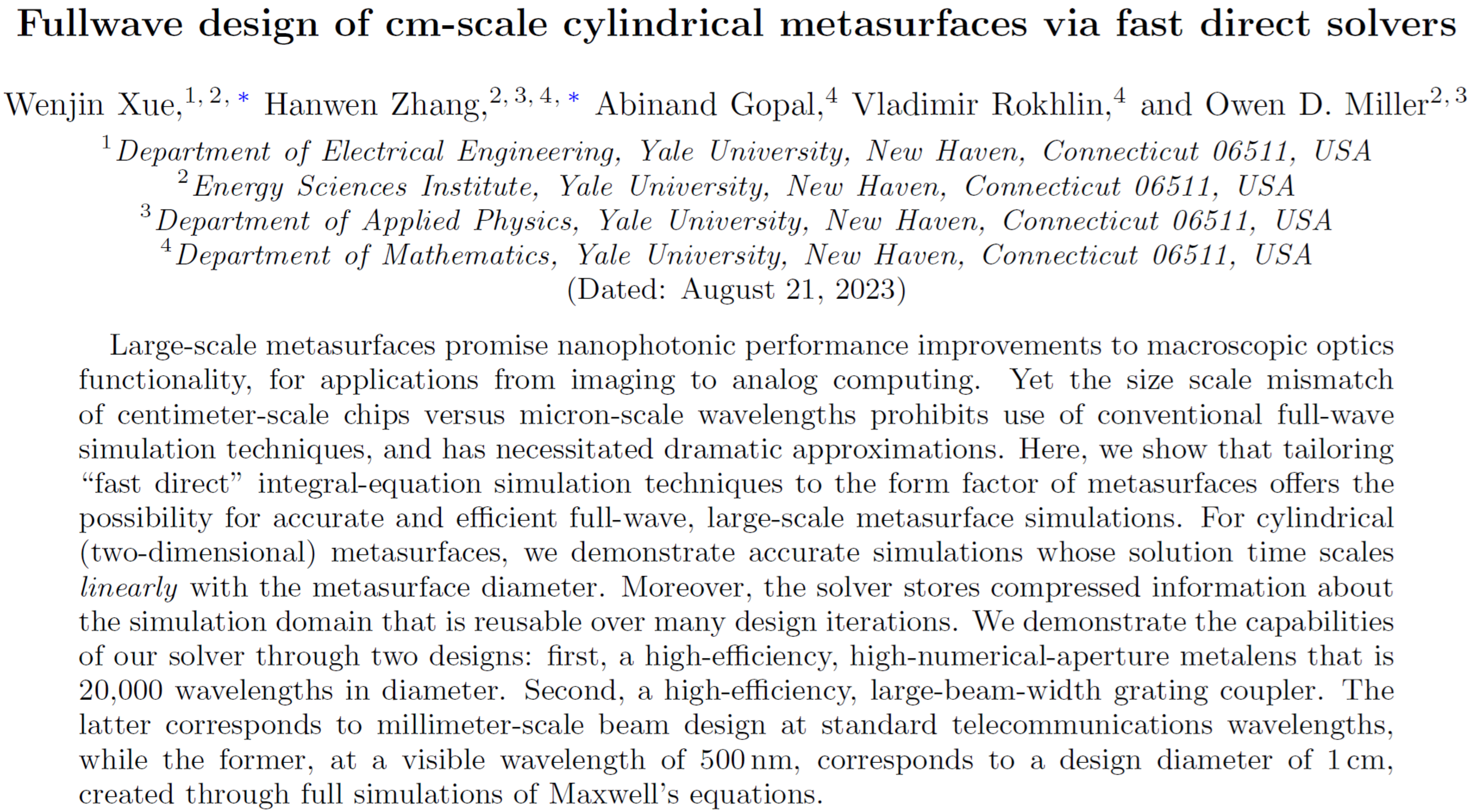
*A. Gopal & P.G. Martinsson, Adv. in Comp. Mathematics, **48**(4), 2022*

Example: Lippmann-Schwinger

Recently extended to time-harmonic Maxwell.

Optimal design problem, so *many* solves required.

Largest experiment involved a lens of diameter 20 000λ.



Fullwave design of cm-scale cylindrical metasurfaces via fast direct solvers

Wenjin Xue,[1, 2, *] Hanwen Zhang,[2, 3, 4, *] Abinand Gopal,[4] Vladimir Rokhlin,[4] and Owen D. Miller[2, 3]

[1] *Department of Electrical Engineering, Yale University, New Haven, Connecticut 06511, USA*
[2] *Energy Sciences Institute, Yale University, New Haven, Connecticut 06511, USA*
[3] *Department of Applied Physics, Yale University, New Haven, Connecticut 06511, USA*
[4] *Department of Mathematics, Yale University, New Haven, Connecticut 06511, USA*

(Dated: August 21, 2023)

Large-scale metasurfaces promise nanophotonic performance improvements to macroscopic optics functionality, for applications from imaging to analog computing. Yet the size scale mismatch of centimeter-scale chips versus micron-scale wavelengths prohibits use of conventional full-wave simulation techniques, and has necessitated dramatic approximations. Here, we show that tailoring "fast direct" integral-equation simulation techniques to the form factor of metasurfaces offers the possibility for accurate and efficient full-wave, large-scale metasurface simulations. For cylindrical (two-dimensional) metasurfaces, we demonstrate accurate simulations whose solution time scales *linearly* with the metasurface diameter. Moreover, the solver stores compressed information about the simulation domain that is reusable over many design iterations. We demonstrate the capabilities of our solver through two designs: first, a high-efficiency, high-numerical-aperture metalens that is 20,000 wavelengths in diameter. Second, a high-efficiency, large-beam-width grating coupler. The latter corresponds to millimeter-scale beam design at standard telecommunications wavelengths, while the former, at a visible wavelength of 500 nm, corresponds to a design diameter of 1 cm, created through full simulations of Maxwell's equations.

Example: Lippmann-Schwinger

Recently extended to time-harmonic Maxwell.

Optimal design problem, so *many* solves required.

Largest experiment involved a lens of diameter 20 000λ.

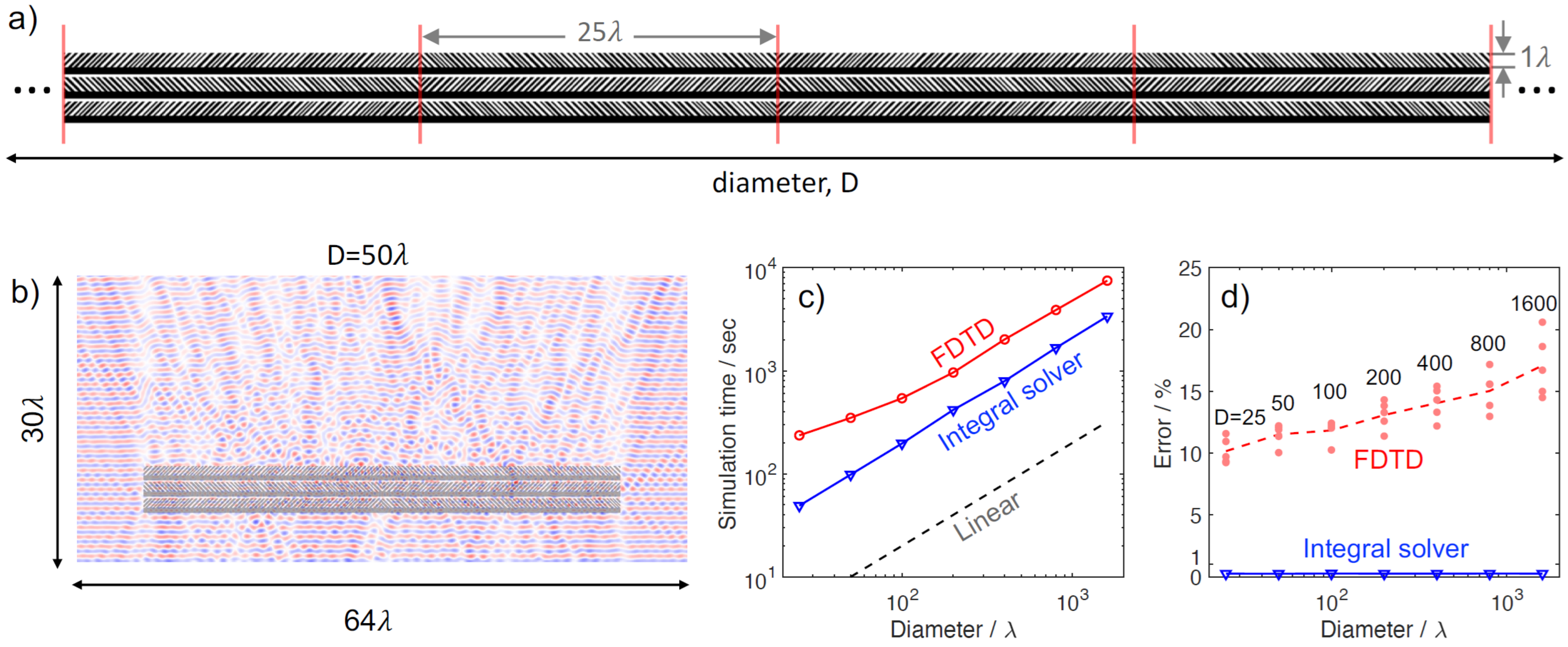


FIG. 3. Comparison between the fast-direct integral-equation method and FDTD for nonlocal metasurfaces. (a) Geomety of the nonlocal metasurfaces: three layers, each generated from a non-symmetric center region comprising two blazed-grating regions (with randomly located slanted pillars) with opposite blazing directions, then adding the same the left/right-pattern outwards, consistently doubling the size. (b) Total electric field in our 50λ-diameter metasurface. (c) Average simulation time over 5 metasurfaces at each diameter, with the simulation resolution fixed, which automatically leads to linear-in-time scaling for FDTD. The integral solver showing linear scaling is a demonstration of the predicted scaling from our compression and proxy-surface techniques. (d) Average simulation error for each technique. Whereas the FDTD solver average error increases from 10% to 18% as the diameter increases (with the largest single-simulation error exceeding 20%), the integral solver maintains less than 1% error.

Key point: Some types of rank structured matrices (HODLR, HSS, HBS, ...) can be inverted using formulas that are:

- Exact.
- Very fast in practice.
- Simple to implement (no recursions, in particular).

Key point: Some types of rank structured matrices (HODLR, HSS, HBS, ...) can be inverted using formulas that are:

- Exact.
- Very fast in practice.
- Simple to implement (no recursions, in particular).

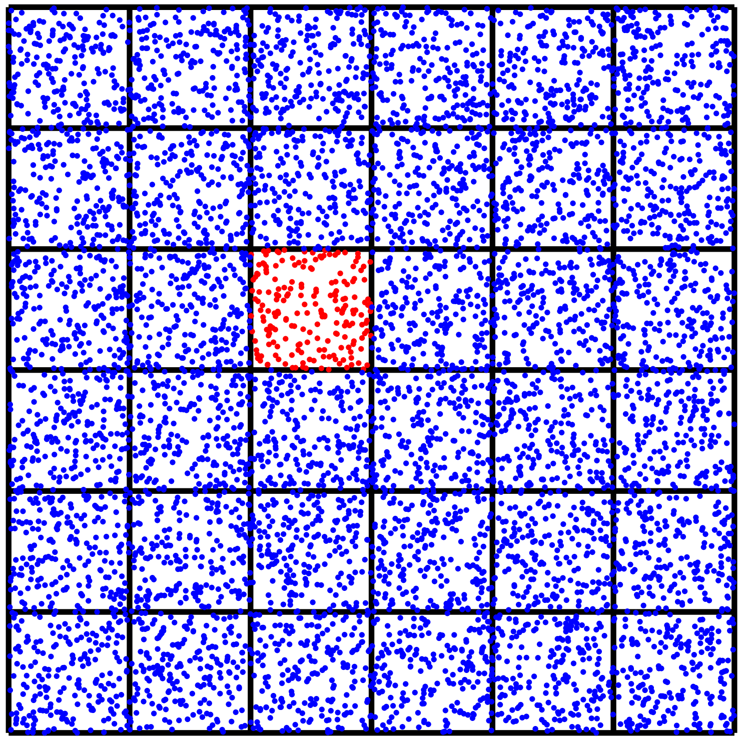
Problem: While these methods work well for 2D problems and medium scale 3D problems, they do not scale correctly in 3D, and quickly become impractical as the problem size grows.

The key problem is that in 3D, the interaction ranks grow with the problem size.

Versions of fast direct solvers: “strong” versus “weak” admissibility

Weak admissibility: Compress directly adjacent patches.

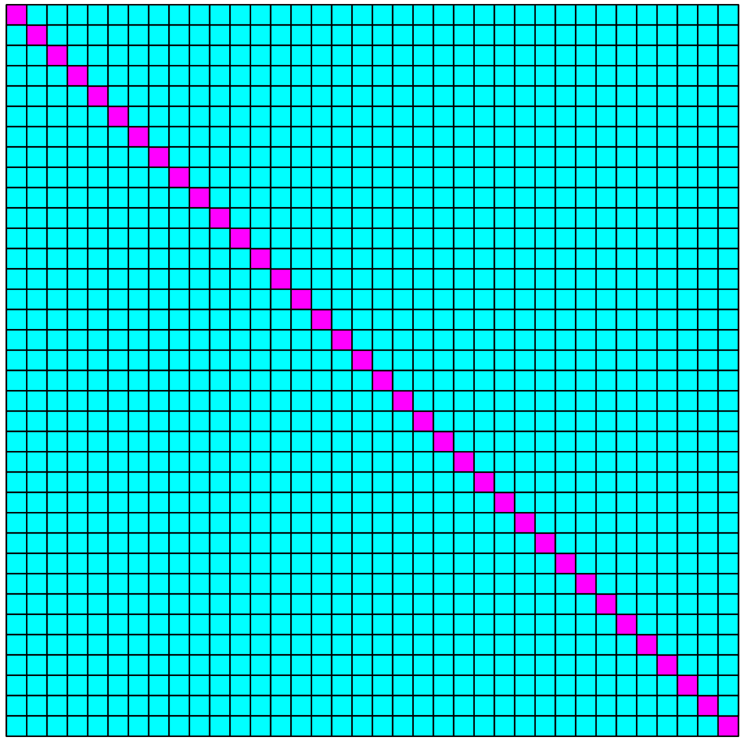
The geometry



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

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

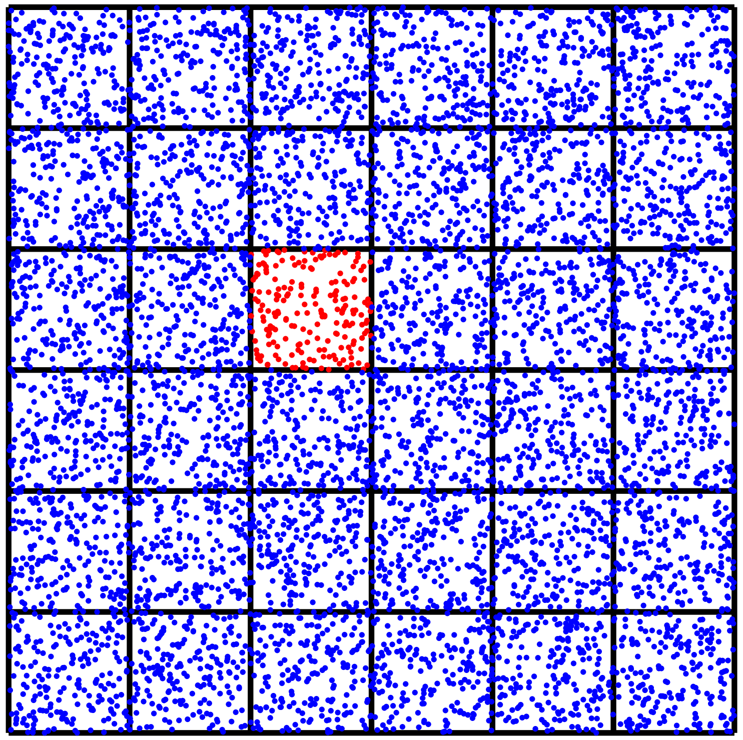
The matrix



Versions of fast direct solvers: “strong” versus “weak” admissibility

Weak admissibility: Compress directly adjacent patches.

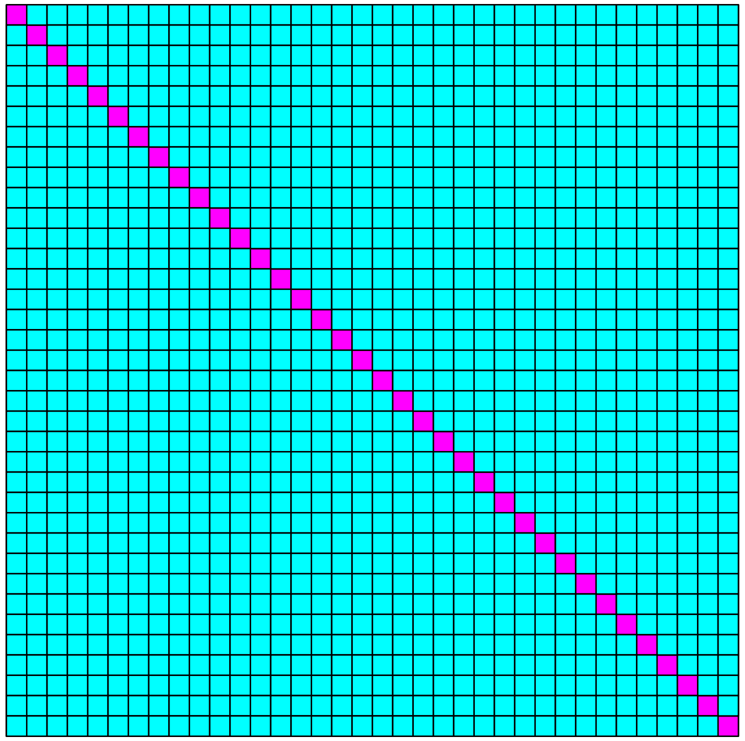
The geometry



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

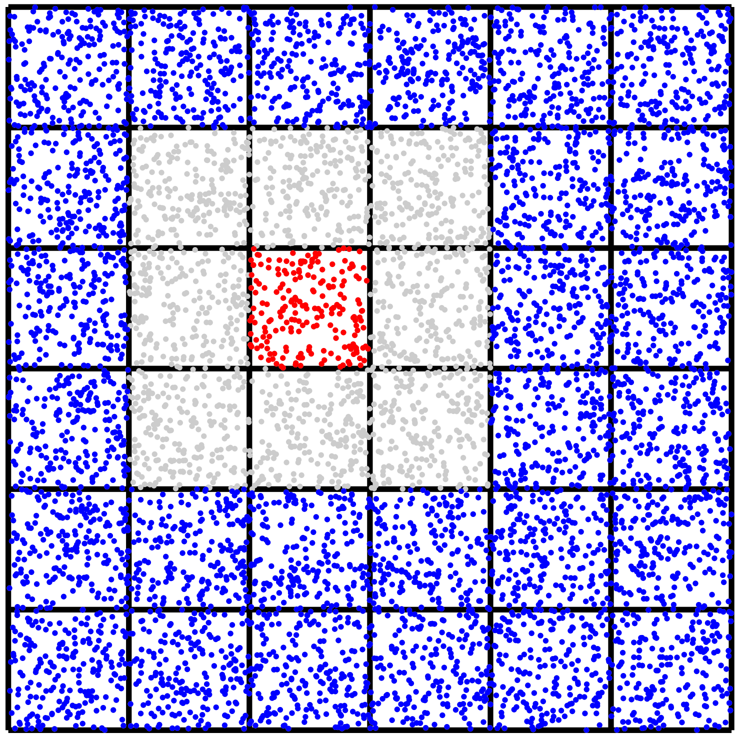
Right: Magenta blocks are dense. Cyan blocks low rank. Many low rank blocks, but high ranks.

The matrix



Strong admissibility: Compress only “far-field” interactions.

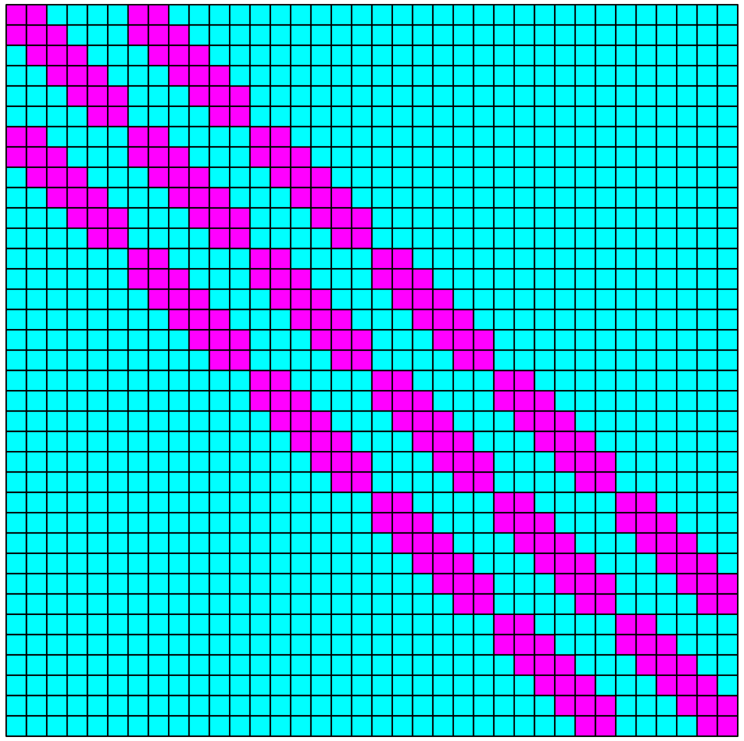
The geometry



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

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

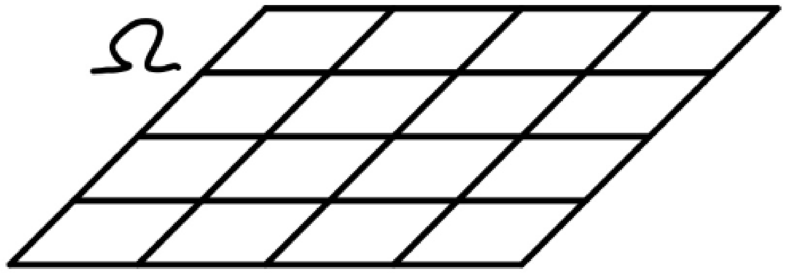
The matrix



Versions of fast direct solvers: “flat” versus “hierarchical” tessellations

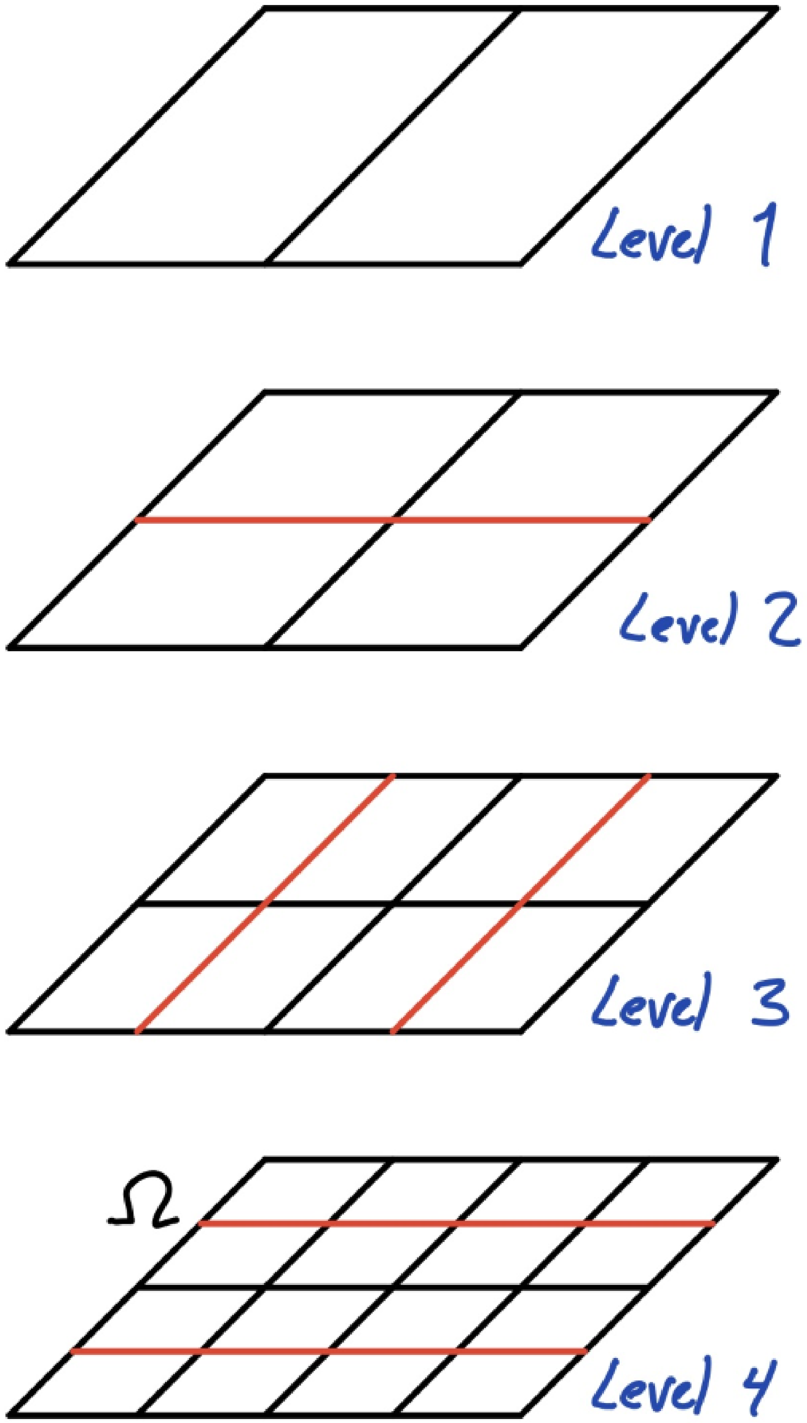
Flat tessellations

Use a single tessellation of the domain.



Hierarchical tessellations

Use a hierarchy of tessellations.

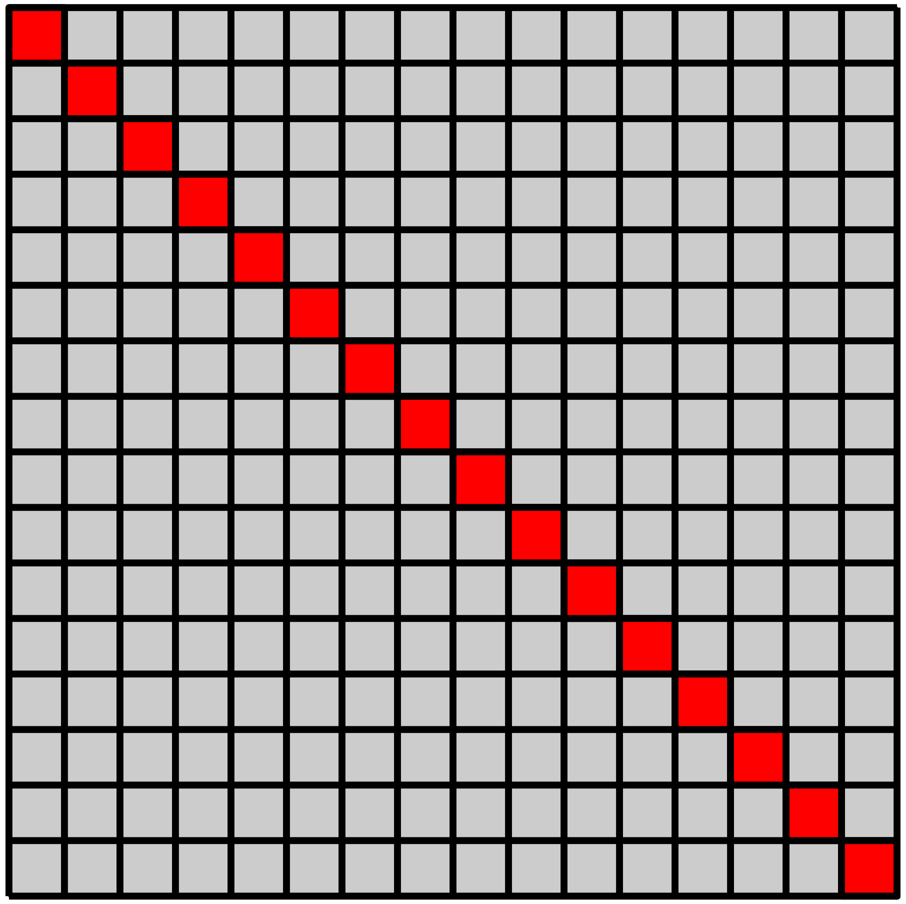


Versions of fast direct solvers: “flat” versus “hierarchical” tessellations

Flat tessellations

Use a single tessellation of the domain.

The resulting matrix:



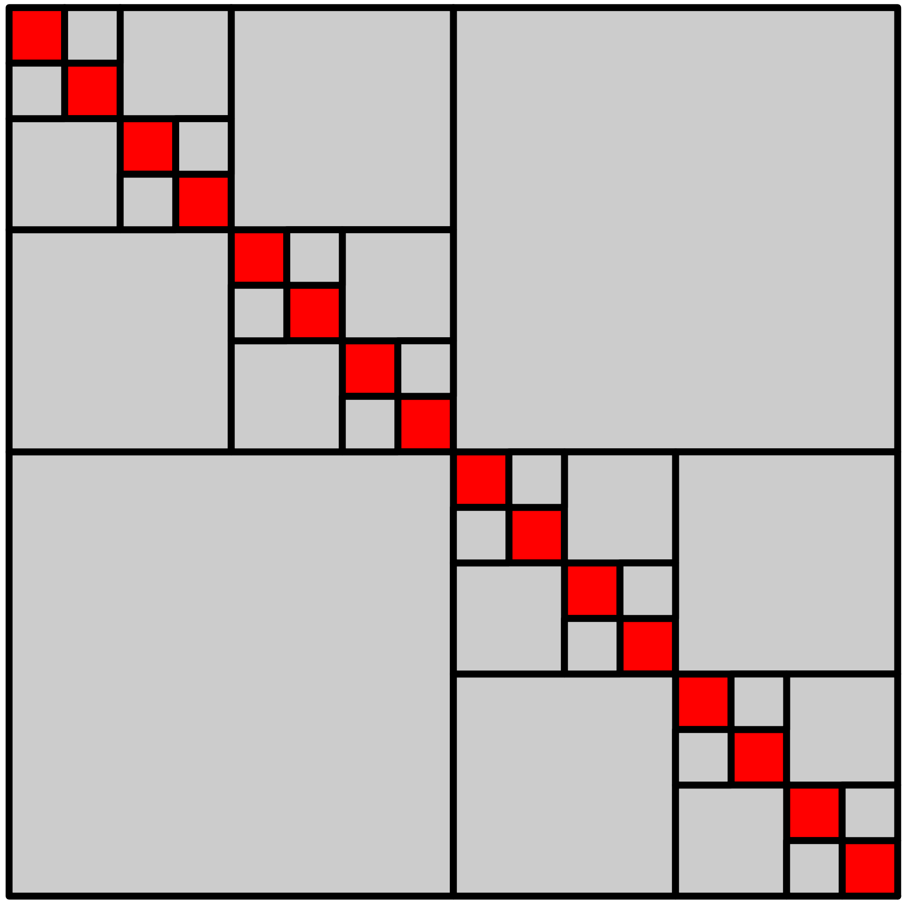
Easy to work with.

Sometimes “good enough”.

Hierarchical tessellations

Use a hierarchy of tessellations.

The resulting matrix:

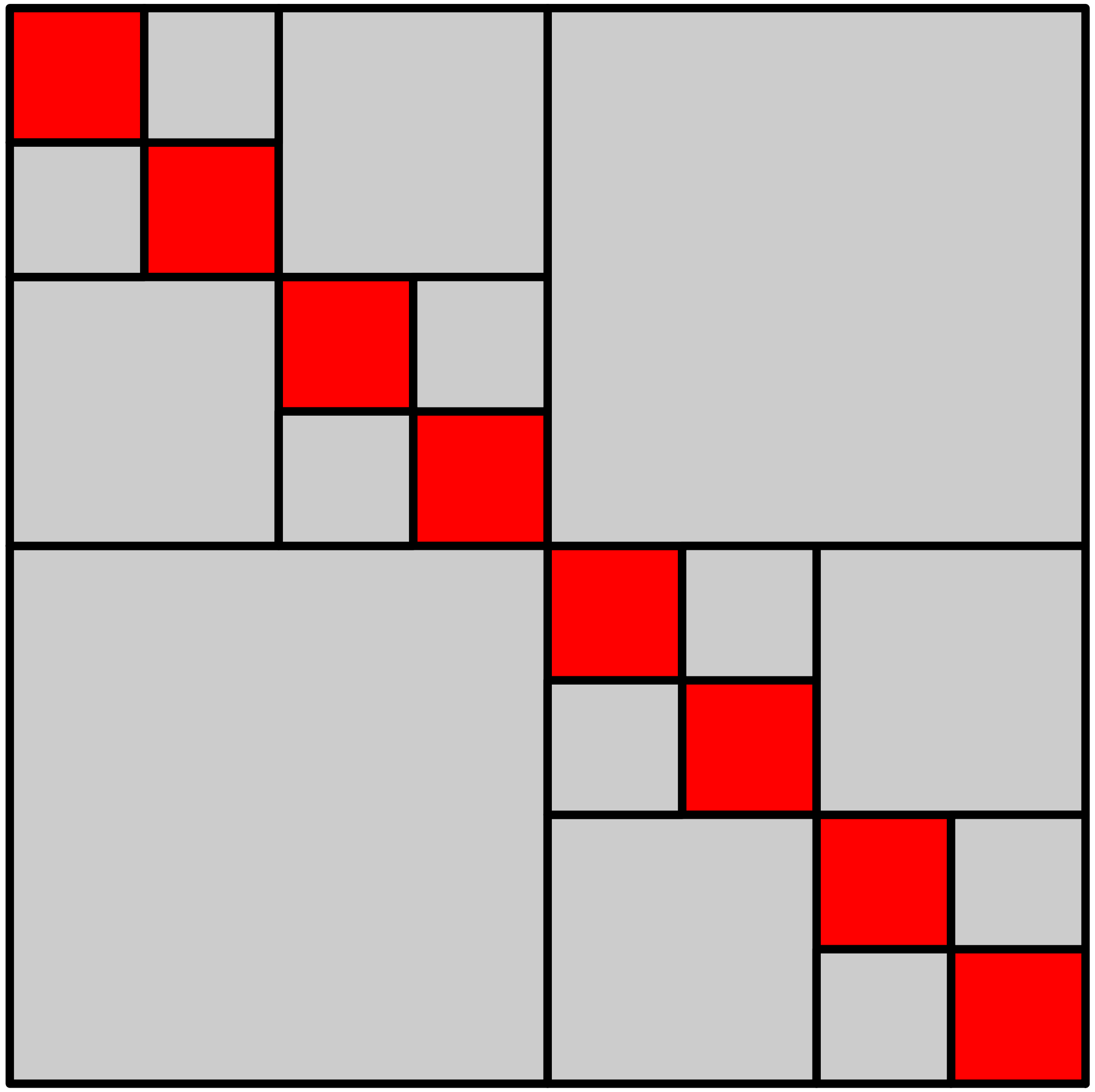


More complicated to code and analyze.

Better asympt. complexity (can be linear).

Versions of fast direct solvers: “nested” versus “general” bases

General bases: Let us consider a basic rank structured matrix:

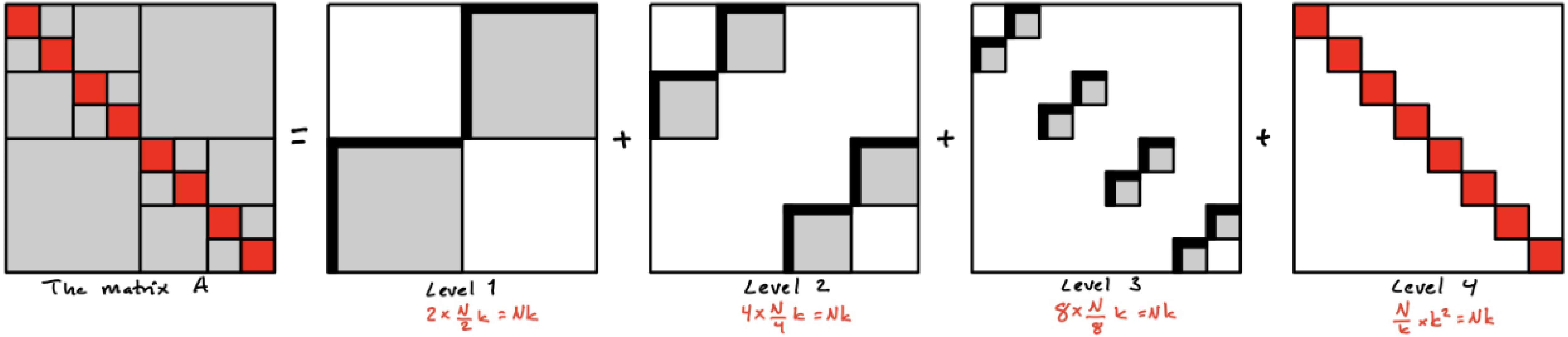


Question: How much storage is required?

Versions of fast direct solvers: “nested” versus “general” bases

General bases: Let us consider a basic rank structured matrix:

Observe that you can view the matrix as a *sum* over different “levels”:



The matrix A
=
Level 1
$2 \times \frac{N}{2} k = Nk$
+
Level 2
$4 \times \frac{N}{4} k = Nk$
+
Level 3
$8 \times \frac{N}{8} k = Nk$
+
Level 4
$\frac{N}{k} \times k^2 = Nk$

Let *k* denote the rank of the off-diagonal blocks.

At each level, the cost to store the factors is $\sim Nk$.

There are $\sim \log(N)$ levels, so total storage $\sim kN \log(N)$.

Versions of fast direct solvers: “nested” versus “general” bases

Nested bases: These were introduced to eliminate log-factors, and improve efficiency.

The idea is to define the low rank factors for the off-diagonal blocks *recursively*

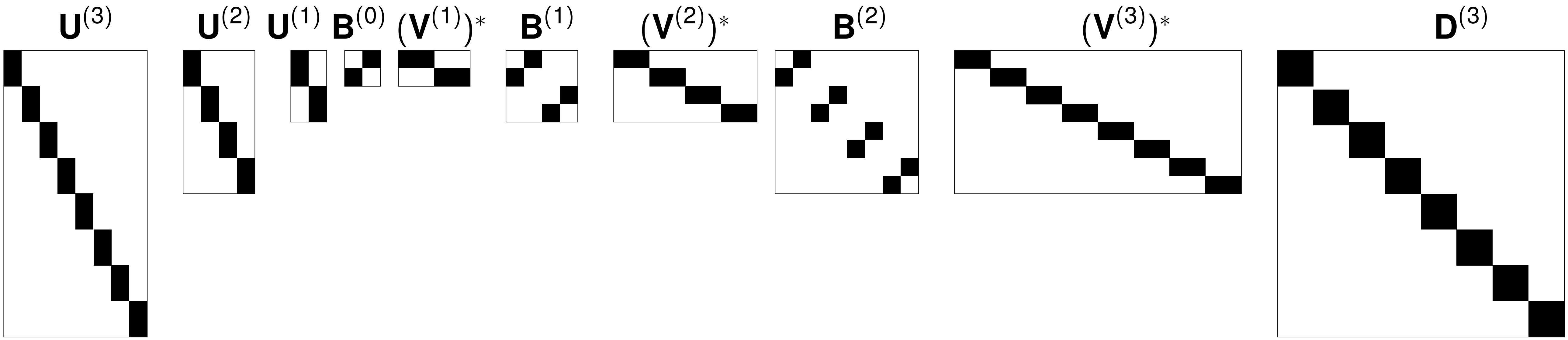
— the bases on one level are defined in terms of the bases on the next finer level.

Formally, this leads to a *multiplicative* representation, rather than an *additive* one.

For instance, it could take the form

$$\mathbf{A} = \mathbf{U}^{(3)}\Big(\mathbf{U}^{(2)}\Big(\mathbf{U}^{(1)}\,\mathbf{B}^{(0)}\,(\mathbf{V}^{(1)})^{*} + \mathbf{B}^{(1)}\Big)(\mathbf{V}^{(2)})^{*} + \mathbf{B}^{(2)}\Big)(\mathbf{V}^{(3)})^{*} + \mathbf{D}^{(3)},$$

where pictorially, the shapes of the factors are as follows:



The cost to store level ℓ is now $2^{-\ell}Nk$ → geometric sum and $O(kN)$ total storage.

Note: The classical Fast Multipole Method relies on nested bases.

This is in contrast to Barnes-Hut which (implicitly) uses general bases.

Versions of fast direct solvers:

We can now loosely organize some common rank-structured matrix “formats”:

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

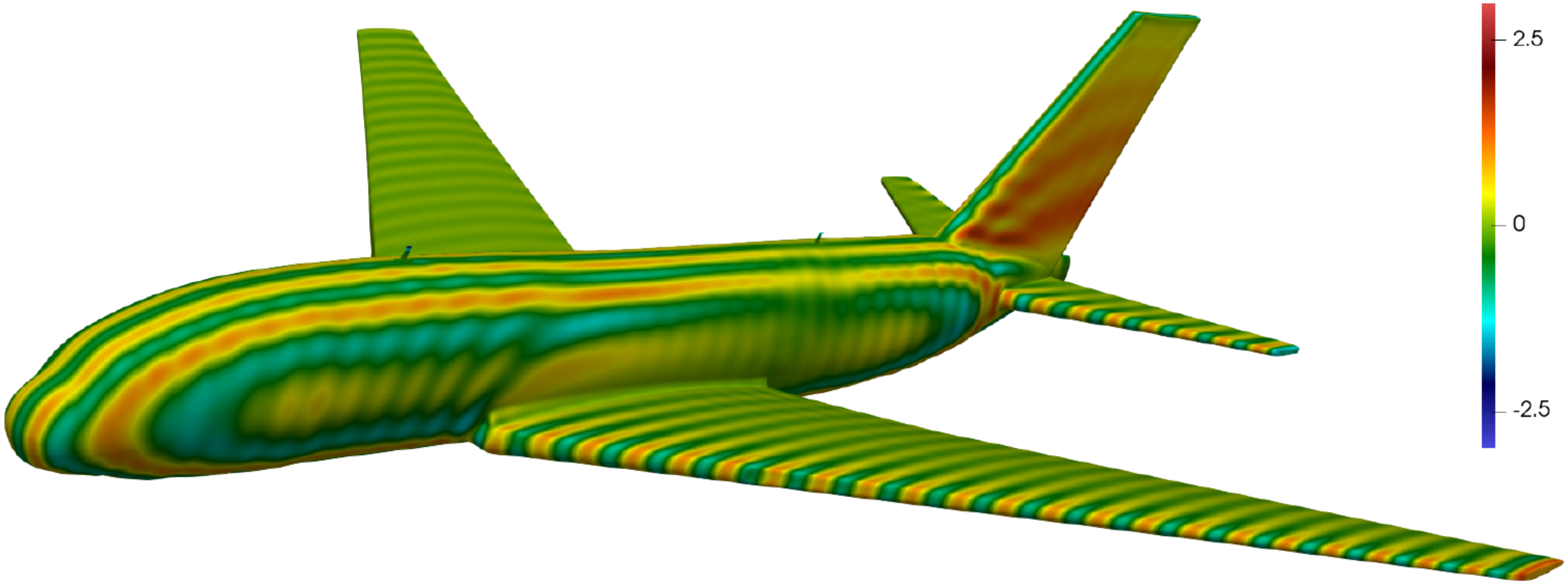
Complexity of implementation *increases* as you go down and to the right in the table.
Asymptotic flop count *decreases* as you go down and to the right in the table.
The higher the dimension, the more complex scheme you need to use.

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

Note: *In principle, the term “$\mathcal{H}$-matrix” is extremely broad — every other format is a special case. However, the table reflects the standard usage of the term.*

Example: Boundary integral equation solver based on recursive skeletonization

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:

$N = 1.2$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: (C. Chen, P.G. Martinsson, M. O'Neil, M. Rachh)

HPC implementation; full parallelization; GPU acceleration; efficient skeletonization.

Versions of fast direct solvers: selection of references

- *$\mathcal{H}$- and $\mathcal{H}^2$-matrices:* Hackbusch (1999); Khoromskij, Hackbusch (2000); Börm, Grasedyck, Hackbusch (2003); . . .
- *Recursive skeletonization:* Lee, Greengard, (1992); Starr, Rokhlin (1994); Michielssen, Boag, Chew (1996); Martinsson, Rokhlin (2005); Greengard, Gueyffier, Martinsson, Rokhlin (2009); Ho, Greengard (2012); Ho, Ying (2016); . . .
- *HSS matrices:* Xia, Chandrasekaran, Gu, and Li (2009); Xia (2012); Wang, Li, Xia, Situ, De Hoop (2013); Xi, Xia (2016); . . .
- *Hierarchically off-diagonal low rank (HODLR) matrices:* Aminfara, Ambikasaran, Darve (2016); Massei, Robol, Kressner (2020); . . .
- *Block low rank (BLR) matrices:* Amestoy, Ashcraft, Boiteau, Buttari, l'Excellent, Weisbecker (2015); Amestoy, Buttari, l'Excellent, Mary (2017); . . .

Survey: Ballani & Kressner (2016).

Monographs: Bebendorf (2008). Börm (2010). Martinsson (2019).

Outline of talk

(1) **The role of global operators in scientific computing.**

(2) **Interaction ranks – why are they small?**

(3) **Introduction to rank structured matrices.**

(4) **Randomized method for compressing global operators.**

“Operator learning”?

Compression of a matrix discretizing a continuum operator

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

Compression of a matrix discretizing a continuum operator

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

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

Compression of a matrix discretizing a continuum operator

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

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

Compression of a matrix discretizing a continuum operator

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

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

In the next several slides, we will discuss two cases:

(1) Warm-up: The global low rank case.

(2) Current research: Full Calderón-Zygmund operator.

Compression of a matrix of global low rank: Randomized SVD

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

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

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m,n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`

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

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ∼] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q’ * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,’econ’)`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

The objective of Stage A is to compute an ON-basis that approximately spans the column space of $\mathbf{A}$. The matrix $\mathbf{Q}$ holds these basis vectors and $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.

Compression of a matrix of global low rank: Randomized SVD

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

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

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m, n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`

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

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

The objective of Stage A is to compute an ON-basis that approximately spans the column space of $\mathbf{A}$. The matrix $\mathbf{Q}$ holds these basis vectors and $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.

Stage B is exact: $\|\mathbf{A} - \mathbf{Q}\underbrace{\mathbf{Q}^*\mathbf{A}}_{=\mathbf{B}}\| = \|\mathbf{A} - \mathbf{Q}\underbrace{\mathbf{B}}_{=\hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*}\| = \|\mathbf{A} - \underbrace{\mathbf{Q}\hat{\mathbf{U}}}_{=\mathbf{U}}\mathbf{D}\mathbf{V}^*\| = \|\mathbf{A} - \mathbf{U}\mathbf{D}\mathbf{V}^*\|$.

Compression of a matrix of global low rank: Randomized SVD

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

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

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m, n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\boldsymbol{\Omega}$. `Omega = randn(n,k)`

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

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\mathrm{ran}(\mathbf{Q}) = \mathrm{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^{*}\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^{*}$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

We claim that the columns of $\mathbf{Y}$ form a good approximate basis for $\mathrm{ran}(\mathbf{A})$.

Observe that $\mathrm{ran}(\mathbf{Y}) \subseteq \mathrm{ran}(\mathbf{A})$ automatically.

Loss of accuracy can happen if $\mathrm{ran}(\mathbf{Y})$ does not capture important directions.

To avoid this, we draw p extra samples, for, say, $p = 5$ or $p = 10$.

Compression of a matrix of global low rank: Randomized SVD

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

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

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m,n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`

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

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^{*}\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^{*}$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

Important: You only need to ensure that you do not undersample.

Over-sampling is unproblematic, since excess data gets “filtered out” in Stage B.

Compression of a matrix of global low rank: Randomized SVD

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

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

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

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

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

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

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

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

Key points:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.

Compression of a matrix of global low rank: Randomized SVD

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

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

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

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

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

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

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

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

Key points:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).

Compression of a matrix of global low rank: Randomized SVD

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

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

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

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

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

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

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

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

Key points:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn\log k)$.

Compression of a matrix of global low rank: Randomized SVD

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

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

```
(1) Draw an $n \times (k+p)$ random matrix $\mathbf{\Omega}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{R}$.
(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^{*}\,\mathbf{A}$.
(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^{*}$.
(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
```

Key points:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn\log k)$.

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

    Cost is now $O(mn)$!

  Practical acceleration is achieved at ordinary matrix sizes.

Compressi

Input: An m $p = 5$).

Output: Ra

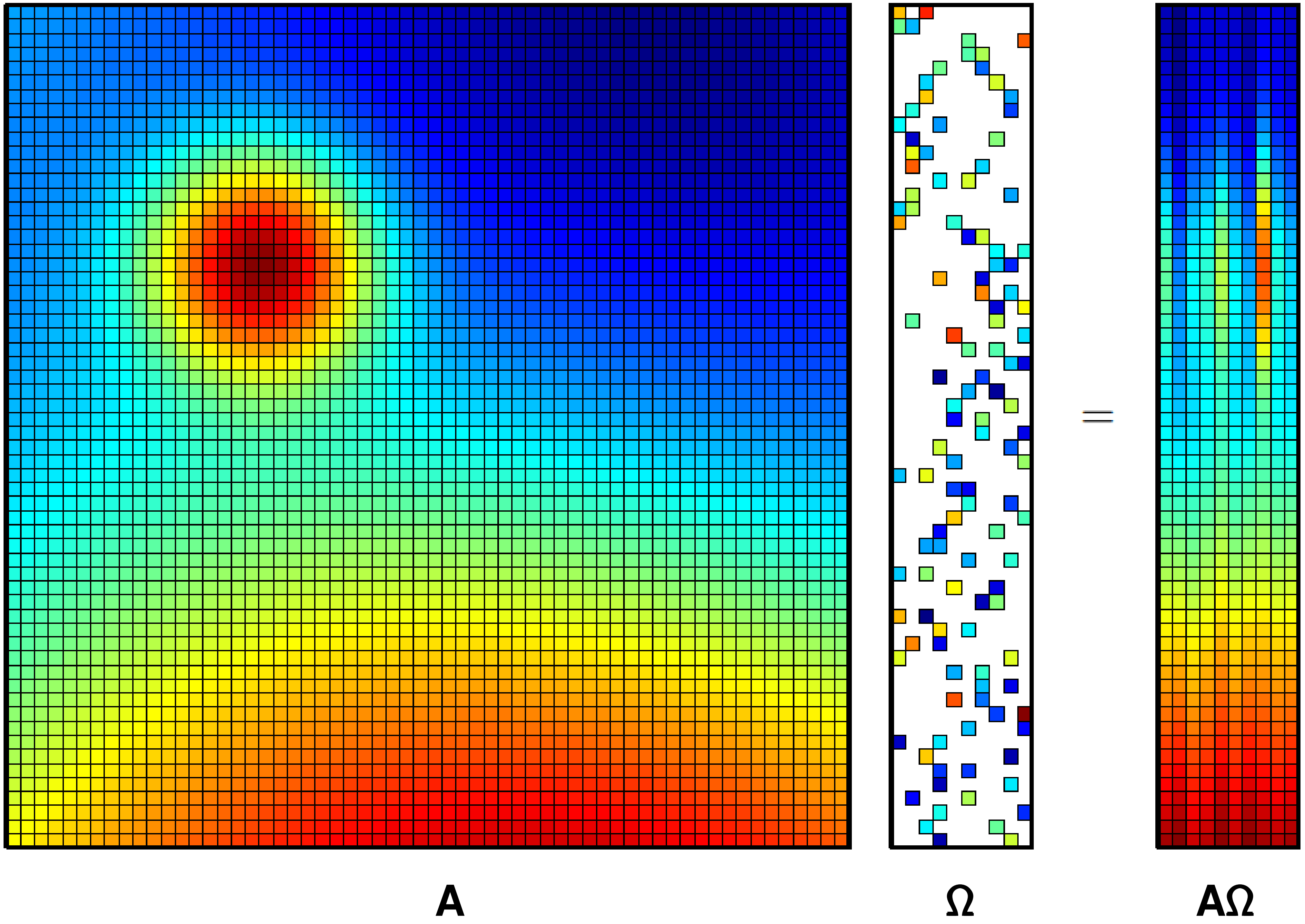
(1) Draw a $\mathbf{Q}^*\,\mathbf{A}$.

(2) Form th $\hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(3) Compu

Key points:

- High pra on.
- Order o
- Highly e
- Conside of a dense n k).



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

Compression of a matrix of global low rank: Randomized SVD

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

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

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

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

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

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

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

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

Key points:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn \log k)$.
- Single pass algorithms have been developed for *streaming environments.*

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

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

Compression of a matrix of global low rank: Randomized SVD

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

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

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

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

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

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

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

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

Key points:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn \log k)$.
- Single pass algorithms have been developed for *streaming environments.*

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

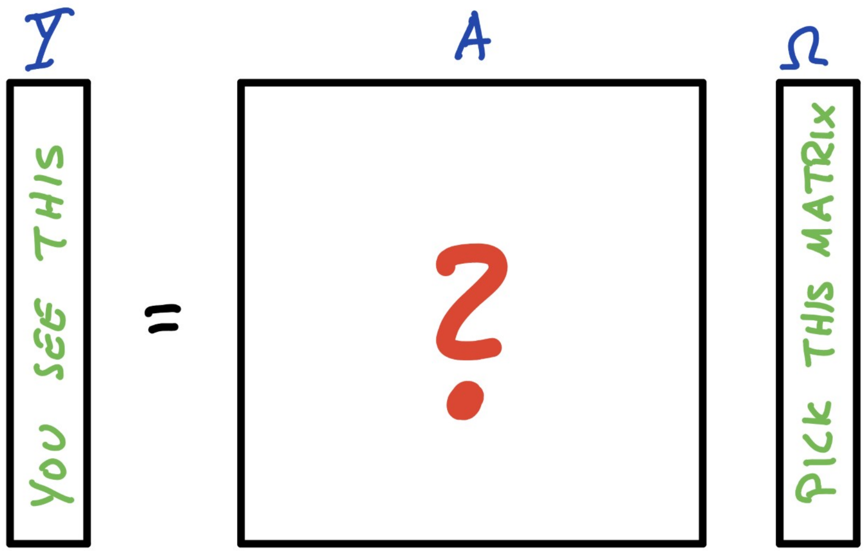
  You cannot store the matrix. *Not possible with deterministic methods!*
- Works exceptionally well for discretized continuum operators due to the *very* fast decay of their singular values. Spectral vs. Frobenius norm.

Compression of a rank-structured matrix

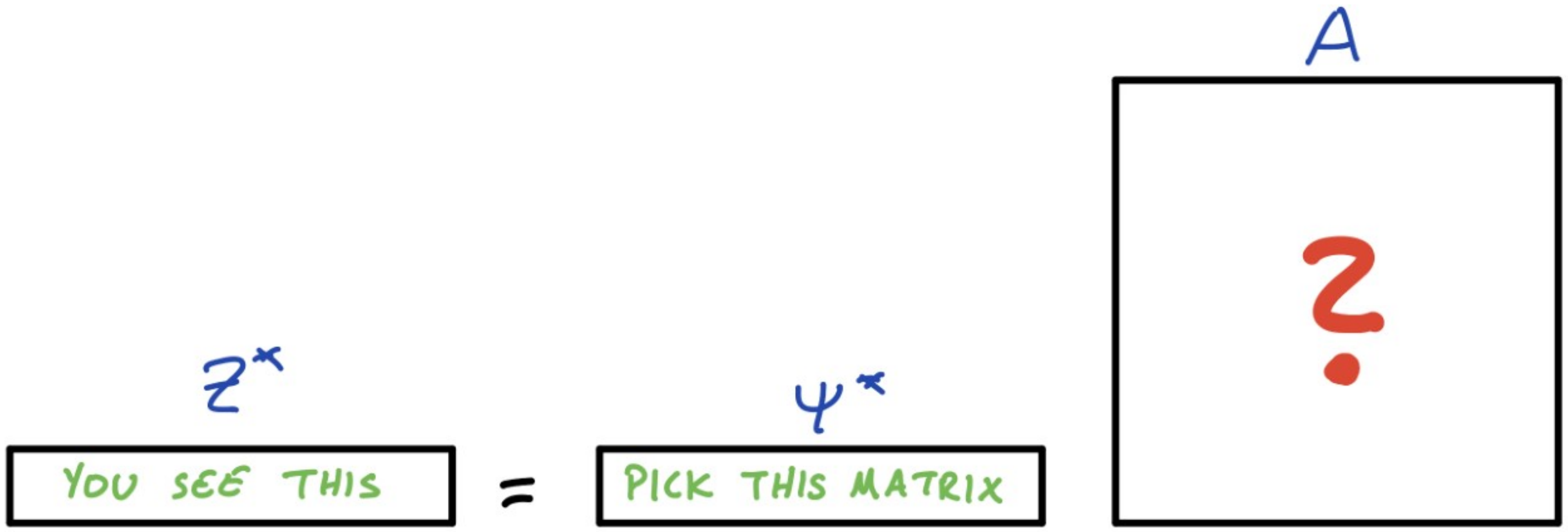
Environment: We are given a rank-structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

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

Sample the column space of the matrix:



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



Compression of a rank-structured matrix

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

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

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

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

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

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

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

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

Compression of a rank-structured matrix

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

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

Why generalize from "global low rank" to "rank-structured":

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

Compression of a rank-structured matrix

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

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

Available techniques for the rank structured case:

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

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

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

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

Compression of a rank-structured matrix

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

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

Available techniques for the rank structured case:

Related work:

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

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

Randomized strong recursive skeletonization: Do inversion and compression simultaneously → order of magnitude reduction in memory requirements. *[Anna Yesypenko PhD thesis, UT-Austin, Nov. 2023. Arxiv:2311.01451]*

Compression of a rank-structured matrix

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

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

Available techniques for the rank structured case:

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

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

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

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

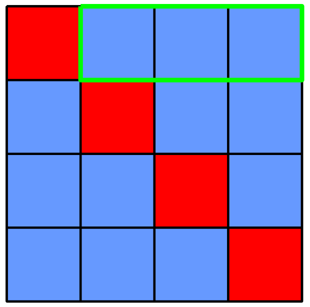
Recent: Fully black box algorithm with true linear complexity and high practical speed:

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

Compression of a rank-structured matrix – A naive approach

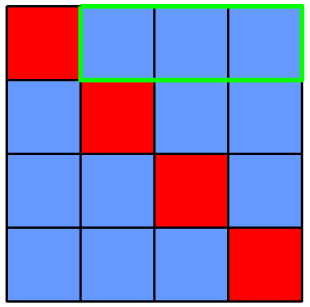
Consider the task of finding a basis matrix $\mathbf{U}_4$ for node 4 using randomized sampling.

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

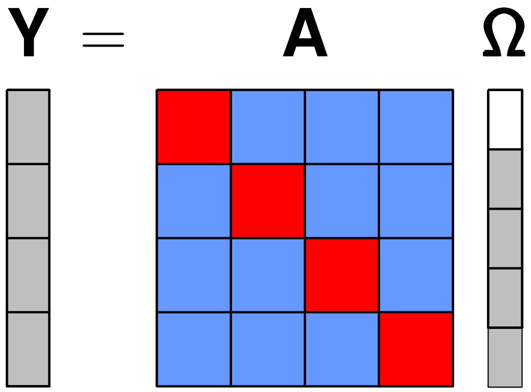


Compression of a rank-structured matrix – A naive approach

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



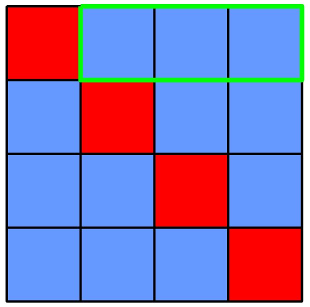
The naive approach is to sample with a random matrix $\boldsymbol{\Omega} \in \mathbb{R}^{N\times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4. Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^c)$.



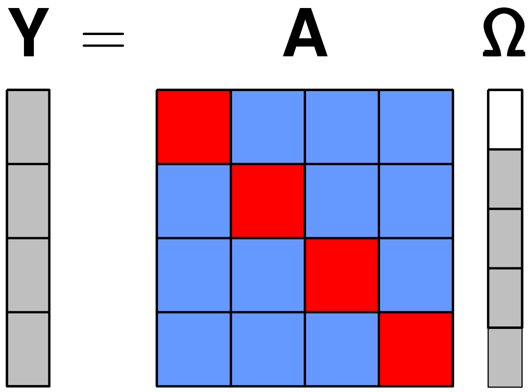
Compression of a rank-structured matrix – A naive approach

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

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



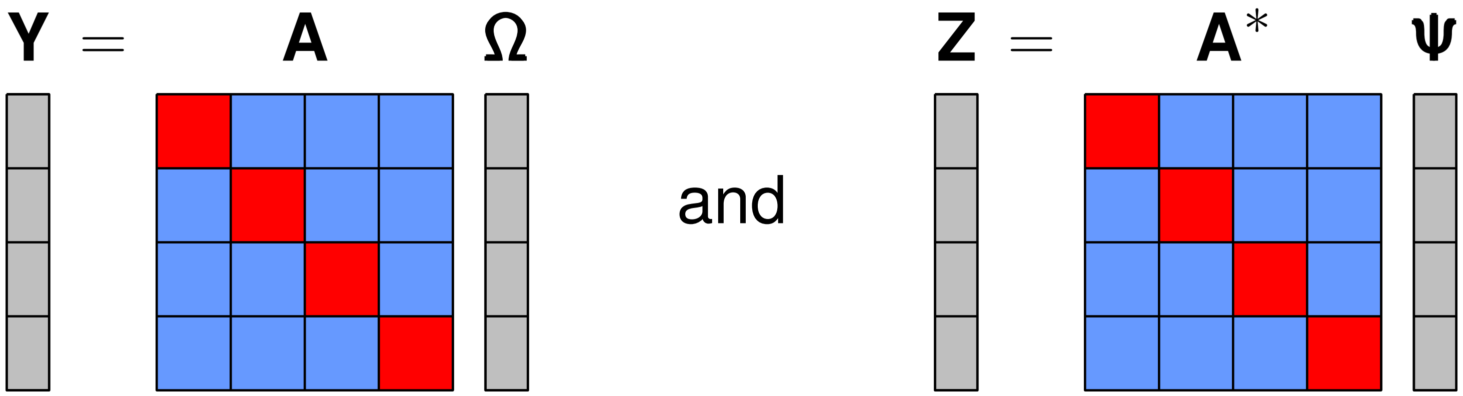
The naive approach is to sample with a random matrix $\mathbf{\Omega} \in \mathbb{R}^{N\times r}$, $r = k + 10$, that has a block of zeros in rows indexed by I_4. Then $\mathbf{Y}(I_4, :)$ will contain a sample of $\mathbf{A}(I_4, I_4^c)$.



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

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

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

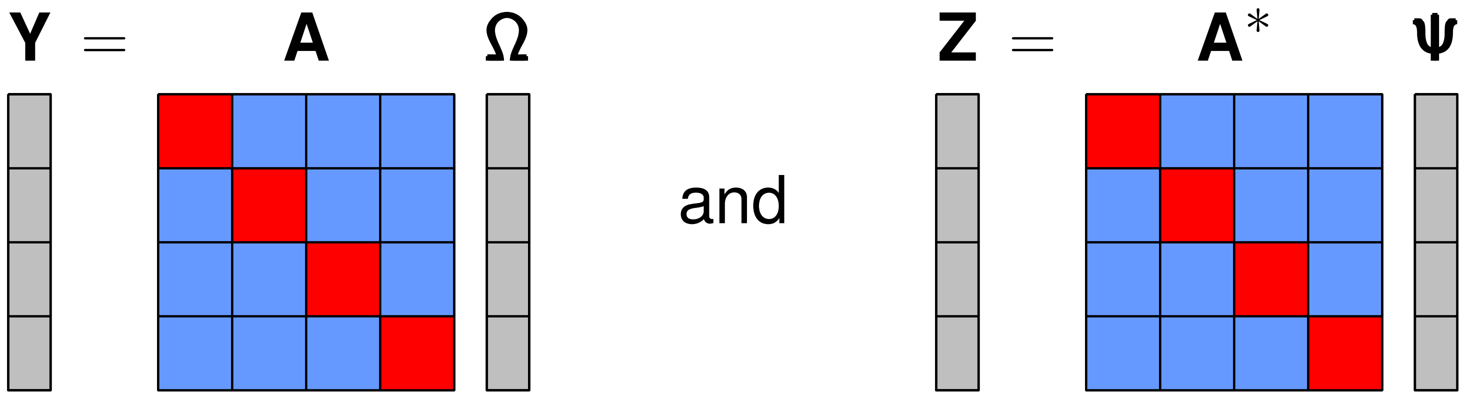


$\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^{*}\,\mathbf{\Psi}$

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

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

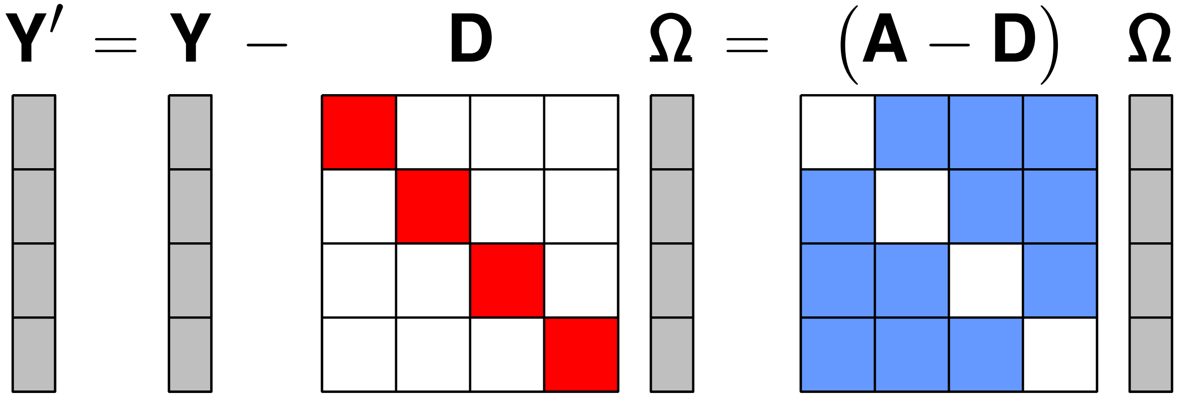
Sample $\mathbf{A}$ with *fixed* dense random matrices $\mathbf{\Omega} \in \mathbb{R}^{N\times r}$ and $\mathbf{\Psi} \in \mathbb{R}^{N\times r}$:



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

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

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



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

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

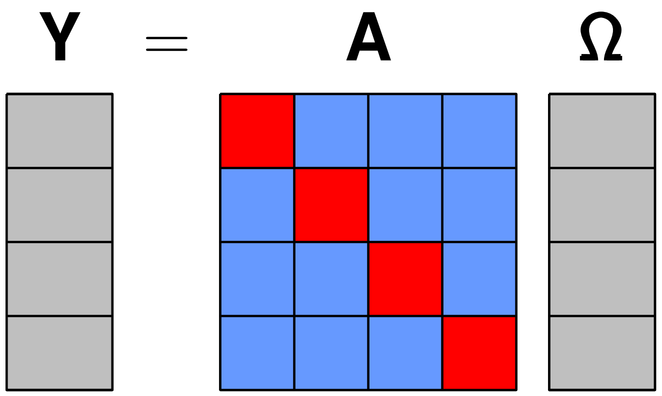
$$\mathbf{A}_{i,j} \approx \mathbf{Y}_i\,\mathbf{B}_{i,j}\,\mathbf{Z}_j^*, \qquad i \neq j,$$

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

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

Compression of a rank-structured matrix – fully black box

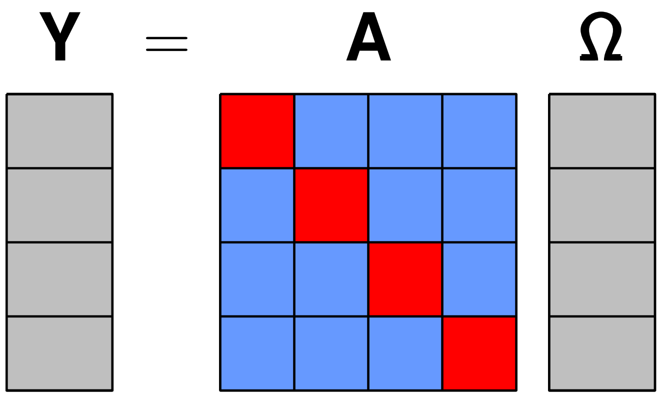
Sample $\mathbf{A}$ with a completely dense random matrix $\mathbf{\Omega} \in \mathbb{R}^{N\times(r+m)}$, where m is the leaf node size. (Think $m \approx 2k$ and $r = k + 10$.)



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

Compression of a rank-structured matrix – fully black box

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



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

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

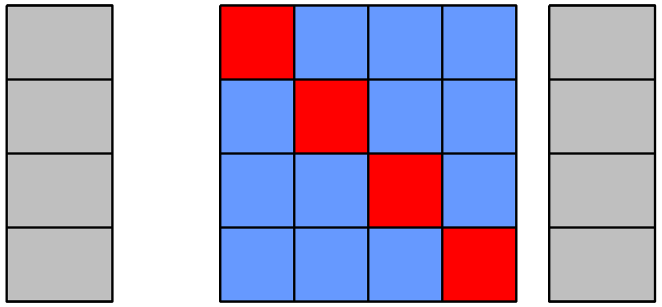
$$\mathbf{Q}_4 = \texttt{nullspace}(\mathbf{\Omega}(I_4, :\,), r)$$

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

Compression of a rank-structured matrix – fully black box

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

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

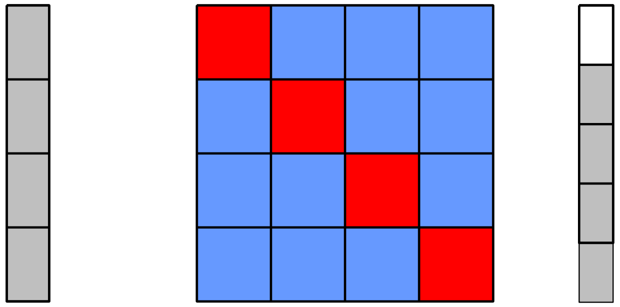


Let us consider the problem of finding a basis matrix $\mathbf{U}_4$ for the block $\mathbf{A}(I_4, I_4^c)$. Since $\mathbf{\Omega}(I_4, :\ )$ is of size $m \times (r+m)$, it has a nullspace of dimension at least r. Let

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

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

$$\mathbf{Y}\mathbf{Q}_4 = \quad \mathbf{A} \quad \mathbf{\Omega}\mathbf{Q}_4.$$



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

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

Compression of a rank-structured matrix – fully black box

- For each leaf node τ, we compute

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

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

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

Compression of a rank-structured matrix – fully black box

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

Steps:

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

```
Compute randomized samples of $\mathbf{A}$ and $\mathbf{A}^*$.
Form Gaussian random random matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ of size $N \times s$.
Multiply $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$.

Compress level by level from finest to coarsest.
for level $\ell = L, L-1, \dots, 0$ do
    for node $\tau$ in level $\ell$ do
        if $\tau$ is a leaf node then
            $\mathbf{\Omega}_\tau = \mathbf{\Omega}(I_\tau, :)$,    $\mathbf{\Psi}_\tau = \mathbf{\Psi}(I_\tau, :)$
            $\mathbf{Y}_\tau = \mathbf{Y}(I_\tau, :)$,    $\mathbf{Z}_\tau = \mathbf{Z}(I_\tau, :)$
        else
            Let $\alpha$ and $\beta$ denote the children of $\tau$.
            $\mathbf{\Omega}_\tau = \begin{bmatrix} \mathbf{V}_\alpha^*\mathbf{\Omega}_\alpha \\ \mathbf{V}_\beta^*\mathbf{\Omega}_\beta \end{bmatrix}$,    $\mathbf{\Psi}_\tau = \begin{bmatrix} \mathbf{U}_\alpha^*\mathbf{\Psi}_\alpha \\ \mathbf{U}_\beta^*\mathbf{\Psi}_\beta \end{bmatrix}$
            $\mathbf{Y}_\tau = \begin{bmatrix} \mathbf{U}_\alpha^*(\mathbf{Y}_\alpha - \mathbf{D}_\alpha\mathbf{\Omega}_\alpha) \\ \mathbf{U}_\beta^*(\mathbf{Y}_\beta - \mathbf{D}_\beta\mathbf{\Omega}_\beta) \end{bmatrix}$,    $\mathbf{Z}_\tau = \begin{bmatrix} \mathbf{V}_\alpha^*(\mathbf{Z}_\alpha - \mathbf{D}_\alpha^*\mathbf{\Psi}_\alpha) \\ \mathbf{V}_\beta^*(\mathbf{Z}_\beta - \mathbf{D}_\beta^*\mathbf{\Psi}_\beta) \end{bmatrix}$

        if level $\ell > 0$ then
            $\mathbf{Q}_\tau = \texttt{nullspace}(\mathbf{\Omega}_\tau, r)$,    $\mathbf{P}_\tau = \texttt{nullspace}(\mathbf{\Psi}_\tau, r)$
            $\mathbf{U}_\tau = \texttt{qr}(\mathbf{Y}_\tau\mathbf{Q}_\tau, r)$,    $\mathbf{V}_\tau = \texttt{qr}(\mathbf{Z}_\tau\mathbf{P}_\tau, r)$
            $\mathbf{D}_\tau = (\mathbf{I} - \mathbf{U}_\tau\mathbf{U}_\tau^*)\mathbf{Y}_\tau\mathbf{\Omega}_\tau^\dagger + \mathbf{U}_\tau\mathbf{U}_\tau^*\left((\mathbf{I} - \mathbf{V}_\tau\mathbf{V}_\tau^*)\mathbf{Z}_\tau\mathbf{\Psi}_\tau^\dagger\right)^*$
        else
            $\mathbf{D}_\tau = \mathbf{Y}_\tau\mathbf{\Omega}_\tau^\dagger$
```

Compression of a rank-structured matrix – Finding D

From the telescoping factorization

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

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

Compression of a rank-structured matrix – Finding D

From the telescoping factorization

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

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

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

Compression of a rank-structured matrix – Finding D

From the telescoping factorization

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

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

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

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

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

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

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

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

and rearrange to obtain

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

Compression of a rank structured matrix: RSRS

Suppose you have extracted samples

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

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

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

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

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

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

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

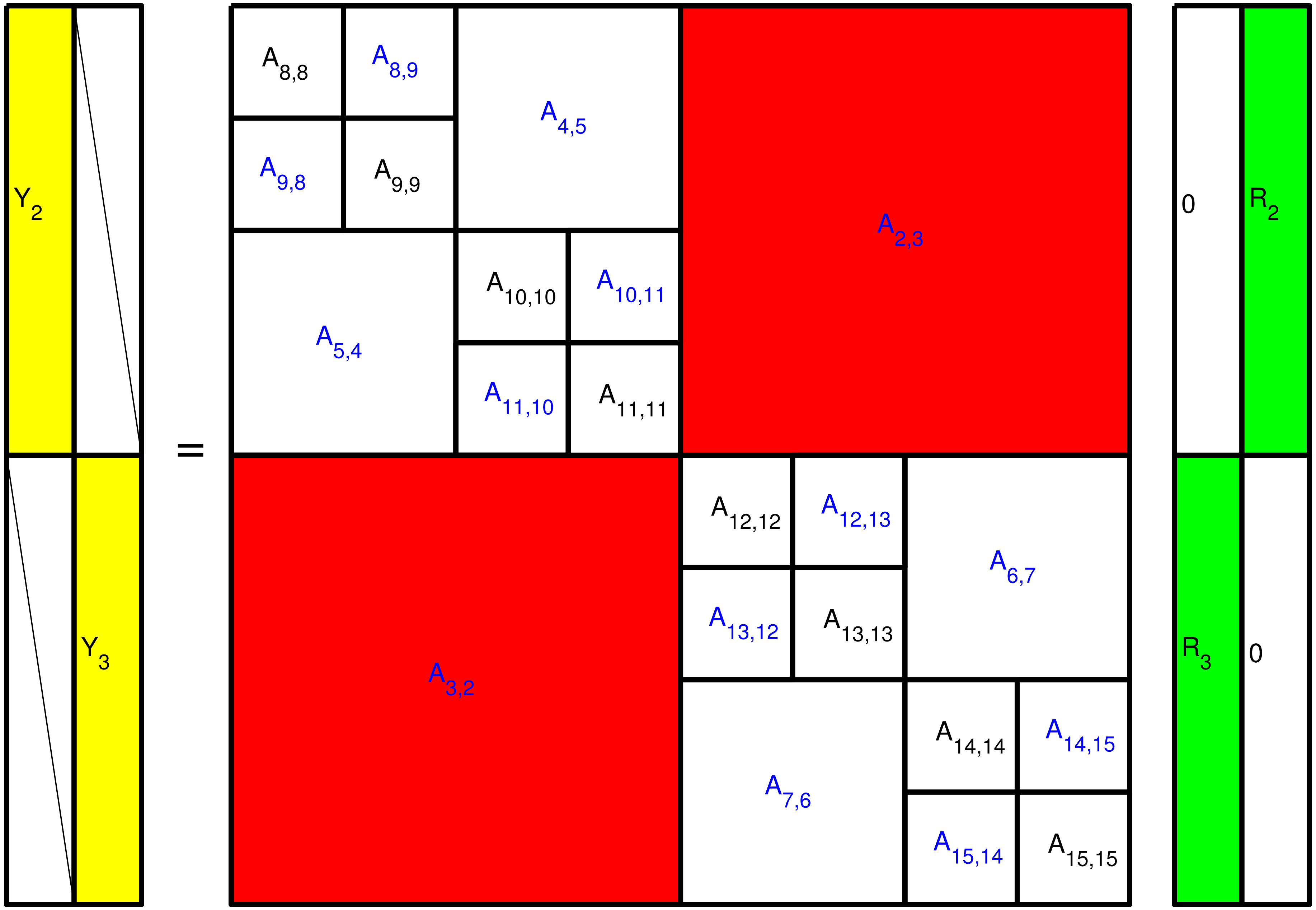
where we defined

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

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

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

The “Peeling algorithm” — level 0



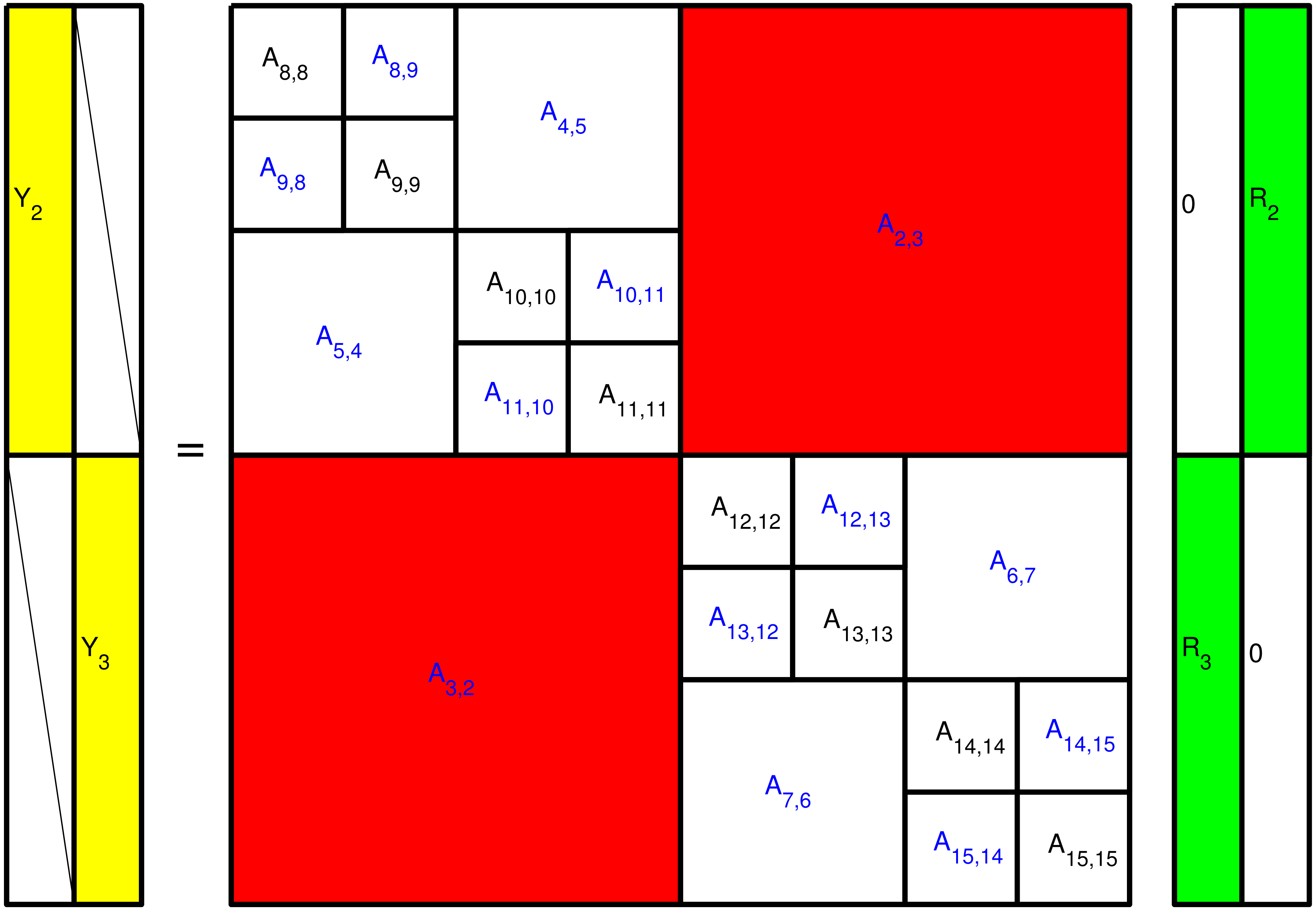
At level 0, we seek to factorize the red blocks.

Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

Now: $\mathbf{Y}_2 = \mathbf{A}_{2,3}\mathbf{R}_3$ and $\mathbf{Y}_3 = \mathbf{A}_{3,2}\mathbf{R}_2$.

The columns of $\mathbf{Y}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\mathcal{U}_\tau = \mathtt{qr}(\mathbf{Y}_\tau)$.

The “Peeling algorithm” — level 0



At level 0, we seek to factorize the red blocks.

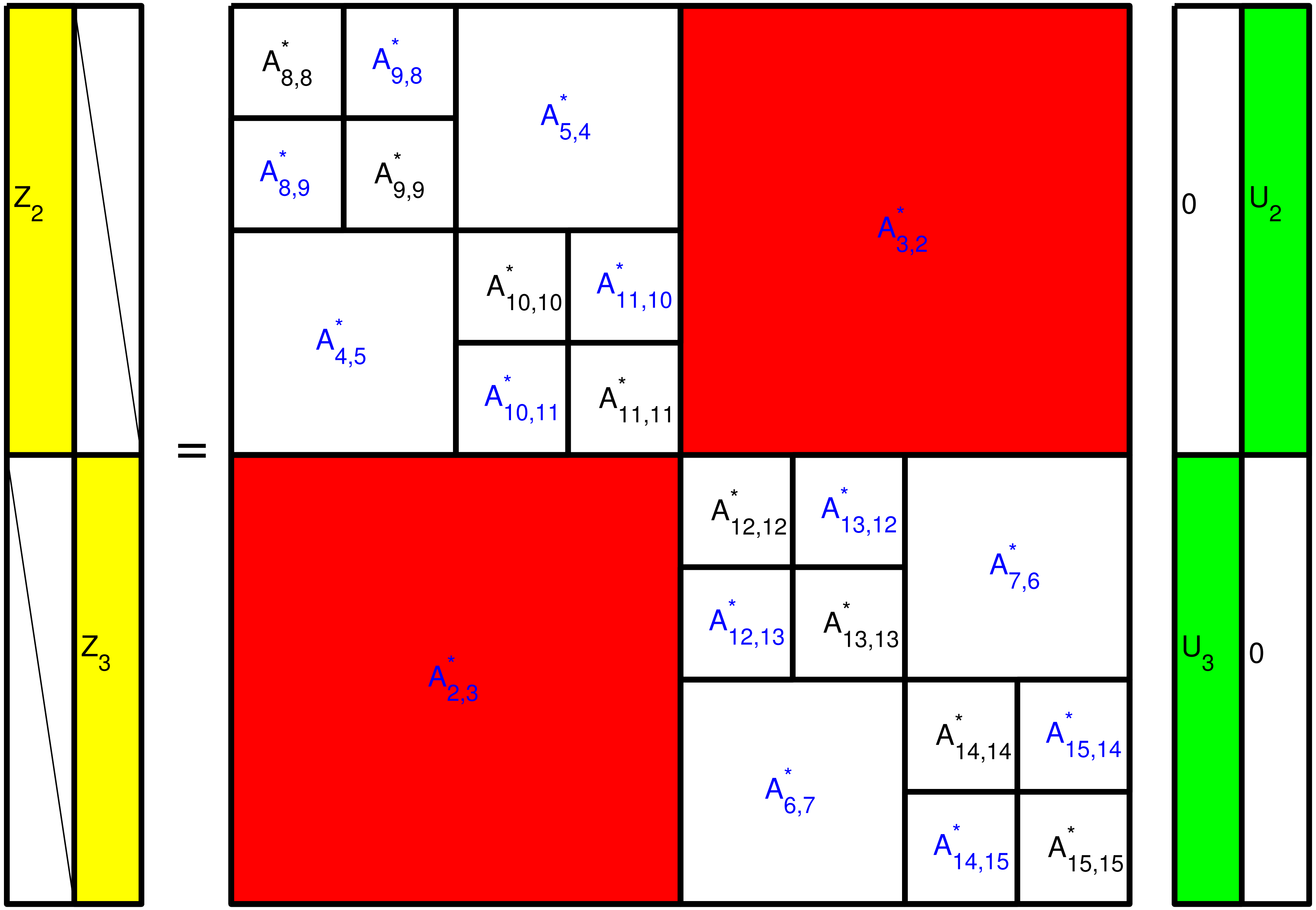
Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

Now: $\mathbf{Y}_2 = \mathbf{A}_{2,3}\mathbf{R}_3$ and $\mathbf{Y}_3 = \mathbf{A}_{3,2}\mathbf{R}_2$.

The columns of $\mathbf{Y}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\boldsymbol{\mathcal{U}}_\tau = \mathtt{qr}(\mathbf{Y}_\tau)$.

How determine $\tilde{\mathbf{A}}_{2,3}$, $\tilde{\mathbf{A}}_{3,2}$, $\boldsymbol{\mathcal{V}}_2$ and $\boldsymbol{\mathcal{V}}_3$?

The “Peeling algorithm” — level 0



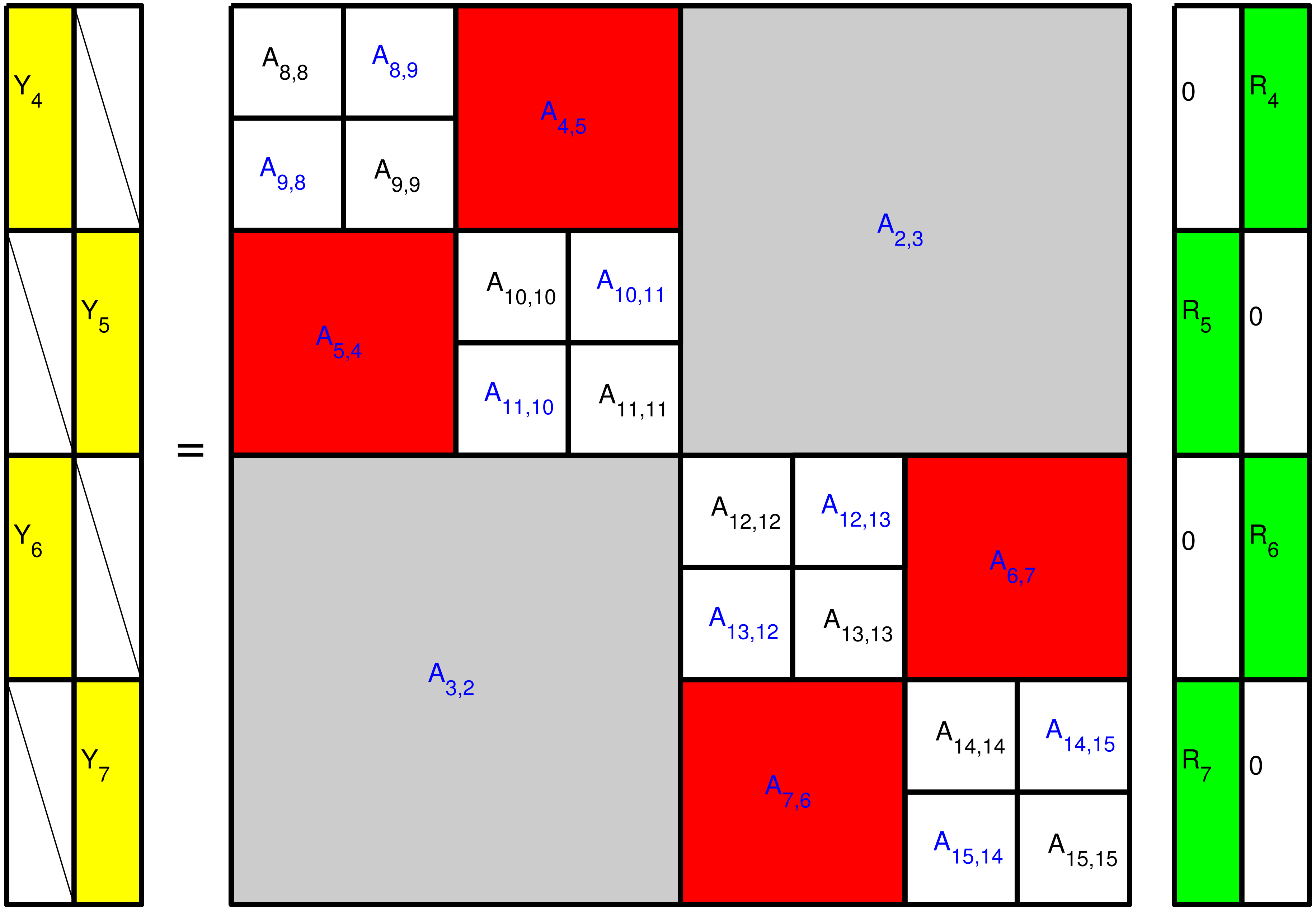
We know $\mathcal{U}_2$ and $\mathcal{U}_3$, and seek to determine $\tilde{\mathbf{A}}_{2,3}$, $\tilde{\mathbf{A}}_{3,2}$, $\mathcal{V}_2$ and $\mathcal{V}_3$.

Put $\mathcal{U}_2$ and $\mathcal{U}_3$ into the probing matrix.

Then: $\mathbf{Z}_2 = \mathbf{A}^*_{3,2}\mathcal{U}_3 = \mathcal{V}_2\mathbf{A}^*_{3,2}$ and $\mathbf{Z}_3 = \mathbf{A}^*_{2,3}\mathcal{U}_2 = \mathcal{V}_3\mathbf{A}^*_{2,3}$.

We then get $[\mathcal{V}_2, \tilde{\mathbf{A}}^*_{3,2}] = \texttt{qr}(\mathbf{Z}_2)$ and $[\mathcal{V}_3, \tilde{\mathbf{A}}^*_{2,3}] = \texttt{qr}(\mathbf{Z}_3)$.

The “Peeling algorithm” — level 1



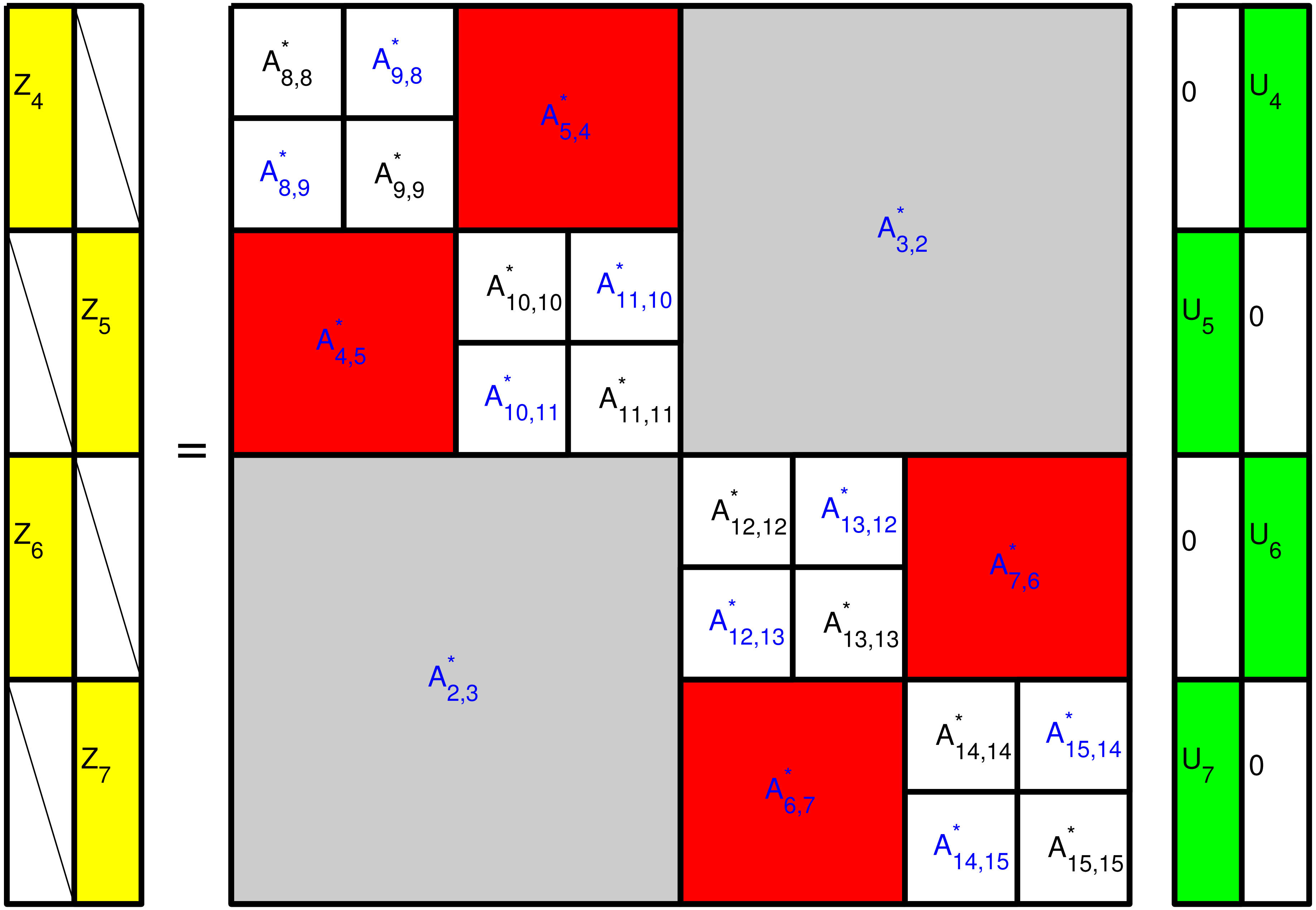
At level 1, we seek to determine the red blocks.

Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

*Subtract the contributions from the **known** gray blocks to get new sample matrices $\mathbf{Z}_\tau$.*

The columns of $\mathbf{Z}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\mathcal{U}_\tau = \texttt{orth}(\mathbf{Y}_\tau, \varepsilon)$.

The “Peeling algorithm” — level 1

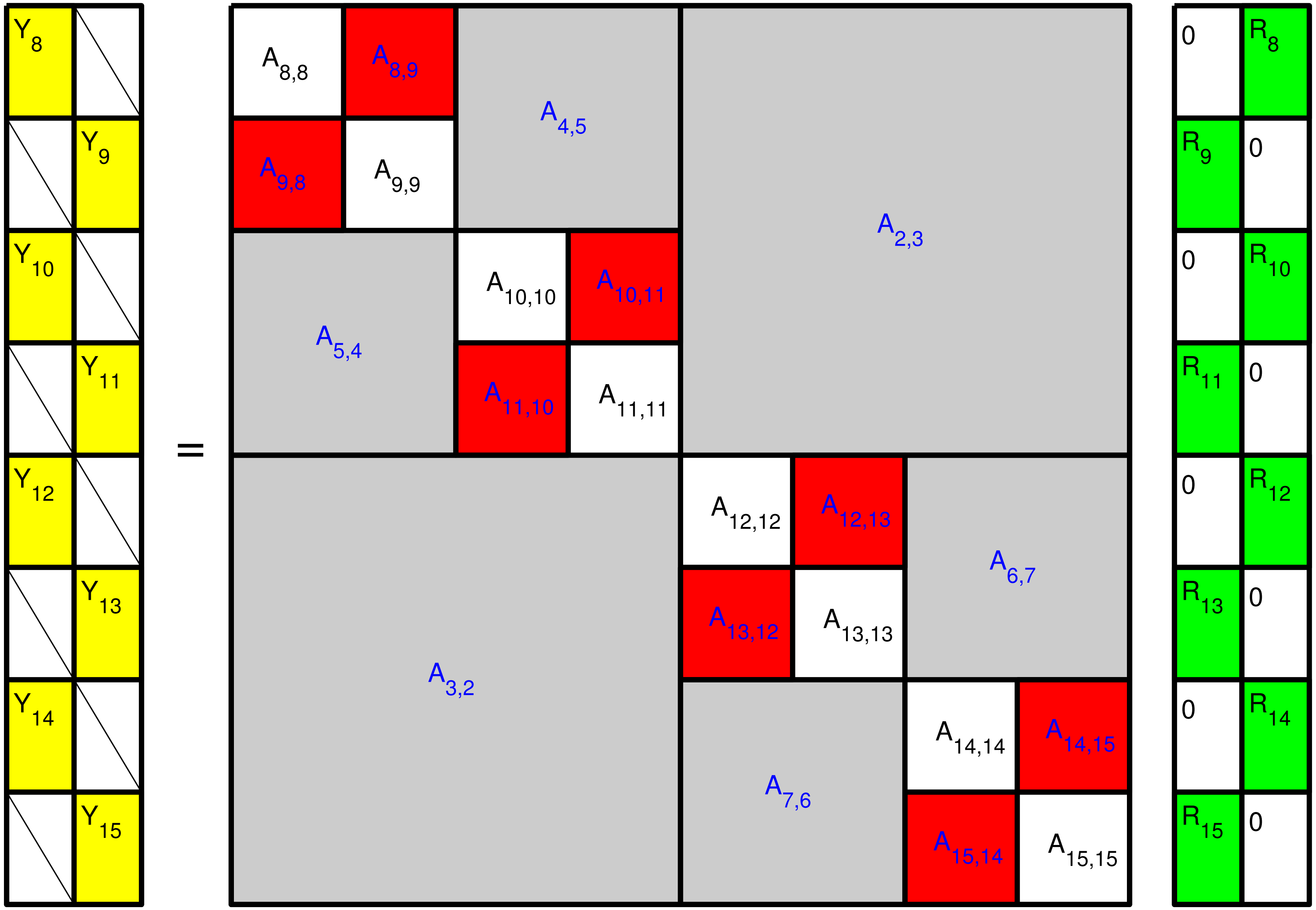


Put the matrices $\{\mathcal{U}_\tau\}_{\tau\text{ is on level 1}}$ into the probing matrix.

Then, e.g., $\mathbf{Z}_4 = \mathbf{A}^*_{5,4}\mathcal{U}_5 = \mathcal{V}_4\mathbf{A}^*_{5,4}$.

We get, e.g., $[\mathcal{V}_4, \tilde{\mathbf{A}}^*_{5,4}] = \mathtt{qr}(\mathbf{Z}_4, 0)$.

The “Peeling algorithm” — level 2



At level 2, we seek to determine the red blocks.

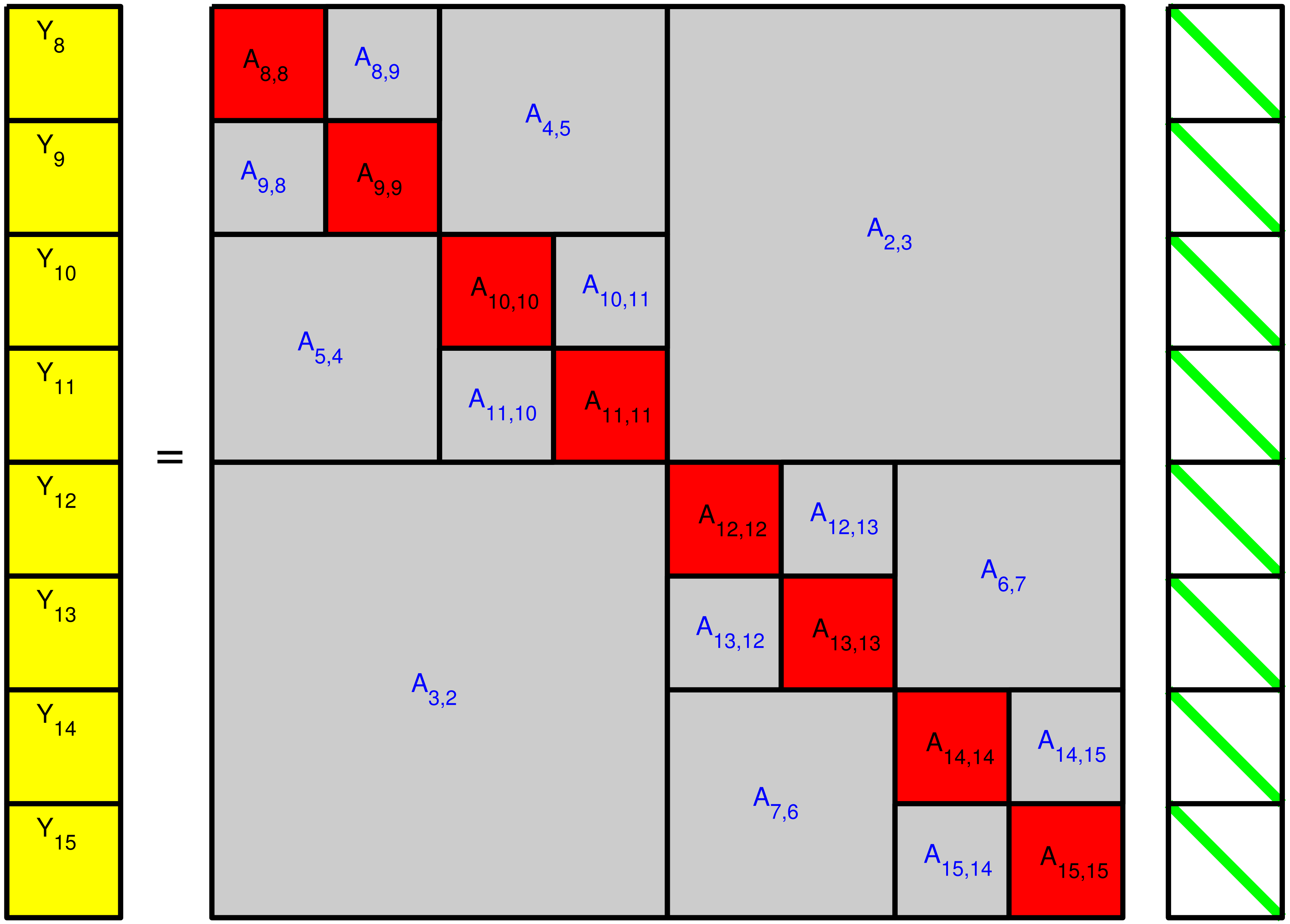
Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

Subtract the contributions from the ***known*** *gray blocks to get new sample matrices* $\mathbf{Z}_\tau$*.*

The columns of $\mathbf{Y}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\mathbf{U}_\tau = \texttt{orth}(\mathbf{Y}_\tau, \varepsilon)$.

By sampling $\mathbf{A}^*$, we determine $\mathbf{V}_\tau$ and $\tilde{\mathbf{A}}_{\tau,\sigma}$.

The “Peeling algorithm” — leaves



Once you reach the finest level, simply use small identity matrices as probing matrices.

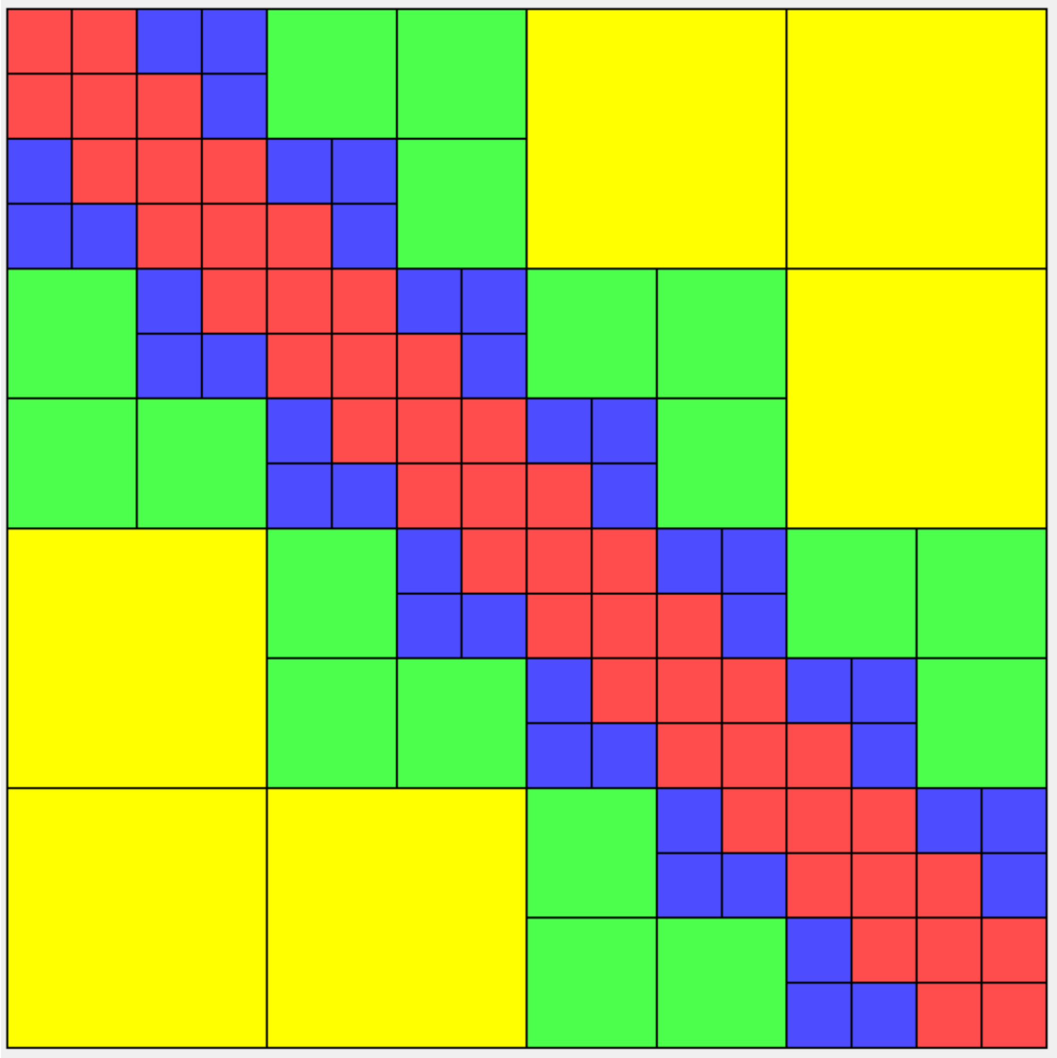
The diagonal blocks $\mathbf{D}_\tau$ are obtained by subtracting contributions from the off-diagonal blocks from $\mathbf{Y}_\tau$.

```
Build compressed representations of all off-diagonal blocks.
loop over levels $\ell = 0 : (L-1)$
    Build the random matrices $\mathbf{R}_1$ and $\mathbf{R}_2$.
    $\mathbf{R}_1 = \mathtt{zeros}(n, r)$
    $\mathbf{R}_2 = \mathtt{zeros}(n, r)$
    loop over boxes $\tau$ on level $\ell$
        Let $\{\alpha, \beta\}$ denote the children of box $\tau$.
        $\mathbf{R}_1(I_\alpha, :) = \mathtt{randn}(n_\alpha, r)$
        $\mathbf{R}_2(I_\beta, :) = \mathtt{randn}(n_\beta, r)$
    end loop
    Apply $\mathbf{A}$ to build the samples for the
    incoming basis matrices.
    $\mathbf{Y}_1 = \mathbf{A}\mathbf{R}_2 - \mathbf{A}^{(\ell)}\mathbf{R}_2$
    $\mathbf{Y}_2 = \mathbf{A}\mathbf{R}_1 - \mathbf{A}^{(\ell)}\mathbf{R}_1$
    Orthonormalize the sample matrices to build
    the incoming basis matrices.
    loop over boxes $\tau$ on level $\ell$
        Let $\{\alpha, \beta\}$ denote the children of box $\tau$.
        $\mathcal{U}_\alpha = \mathtt{qr}(\mathbf{Y}_1(I_\alpha, :))$.
        $\mathcal{U}_\beta = \mathtt{qr}(\mathbf{Y}_2(I_\beta, :))$.
        $\mathbf{R}_1(I_\alpha, :) = \mathcal{U}_\alpha$
        $\mathbf{R}_2(I_\beta, :) = \mathcal{U}_\beta$
    end loop
    Apply $\mathbf{A}^*$ to build the samples for the outgoing basis matrices.
    $\mathbf{Z}_1 = \mathbf{A}^*\mathbf{R}_2 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{R}_2$
    $\mathbf{Z}_2 = \mathbf{A}^*\mathbf{R}_1 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{R}_1$
    Take local SVDs to build incoming basis matrices and
    sibling interaction matrices.
    loop over boxes $\tau$ on level $\ell$
        Let $\{\alpha, \beta\}$ denote the children of box $\tau$.
        $[\mathcal{V}_\alpha, \mathbf{B}_{\beta\alpha}, \hat{\mathbf{U}}_\beta] = \mathtt{svd}(\mathbf{Z}_1(I_\alpha, :), \varepsilon)$.
        $[\mathcal{V}_\beta, \mathbf{B}_{\alpha\beta}, \hat{\mathbf{V}}_\alpha] = \mathtt{svd}(\mathbf{Z}_2(I_\beta, :), \varepsilon)$.
        $\mathcal{U}_\beta \leftarrow \mathcal{U}_\beta \hat{\mathbf{U}}_\beta$.
        $\mathcal{U}_\alpha \leftarrow \mathcal{U}_\alpha \hat{\mathbf{U}}_\alpha$.
    end loop
end loop
Extract the diagonal matrices.
$n_{\max} = \max\{n_\tau : \tau \text{ is a leaf}\}$
$\mathbf{R} = \mathtt{zeros}(N, n_{\max})$
loop over leaf boxes $\tau$
    $\mathbf{R}(I_\tau, 1 : n_\tau) = \mathtt{eye}(n_\tau)$.
end loop
$\mathbf{Y} = \mathbf{A}\mathbf{R} - \mathbf{A}^{(L)}\mathbf{R}$
loop over leaf boxes $\tau$
    $\mathbf{D}_\tau = \mathbf{Y}(I_\tau, 1 : n_\tau)$.
end loop
```

The matrix $\mathbf{A}^{(\ell)}$ consists of all blocks on level ℓ or coarser.

Complication 1: Strong admissibility

Recall that in many applications, we want to avoid compressing boxes that are directly adjacent. This leads to a matrix patterned as follows:

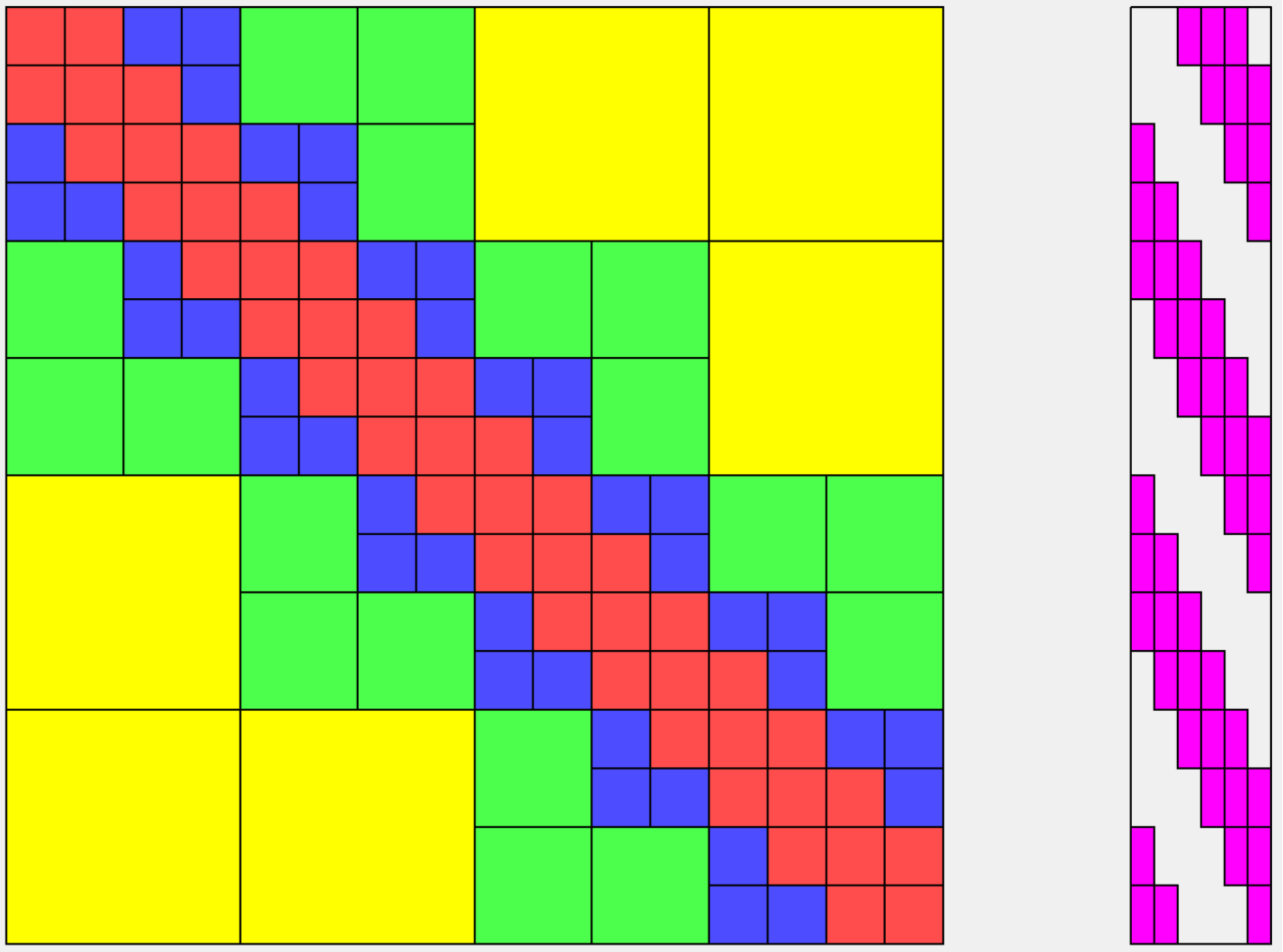


Low rank on level 2. Low rank on level 3. Low rank on level 4. Dense blocks.

Say we know the green and yellow blocks, and seek to compress the blue blocks.

Complication 1: Strong admissibility

Recall that in many applications, we want to avoid compressing boxes that are directly adjacent. This leads to a matrix patterned as follows:



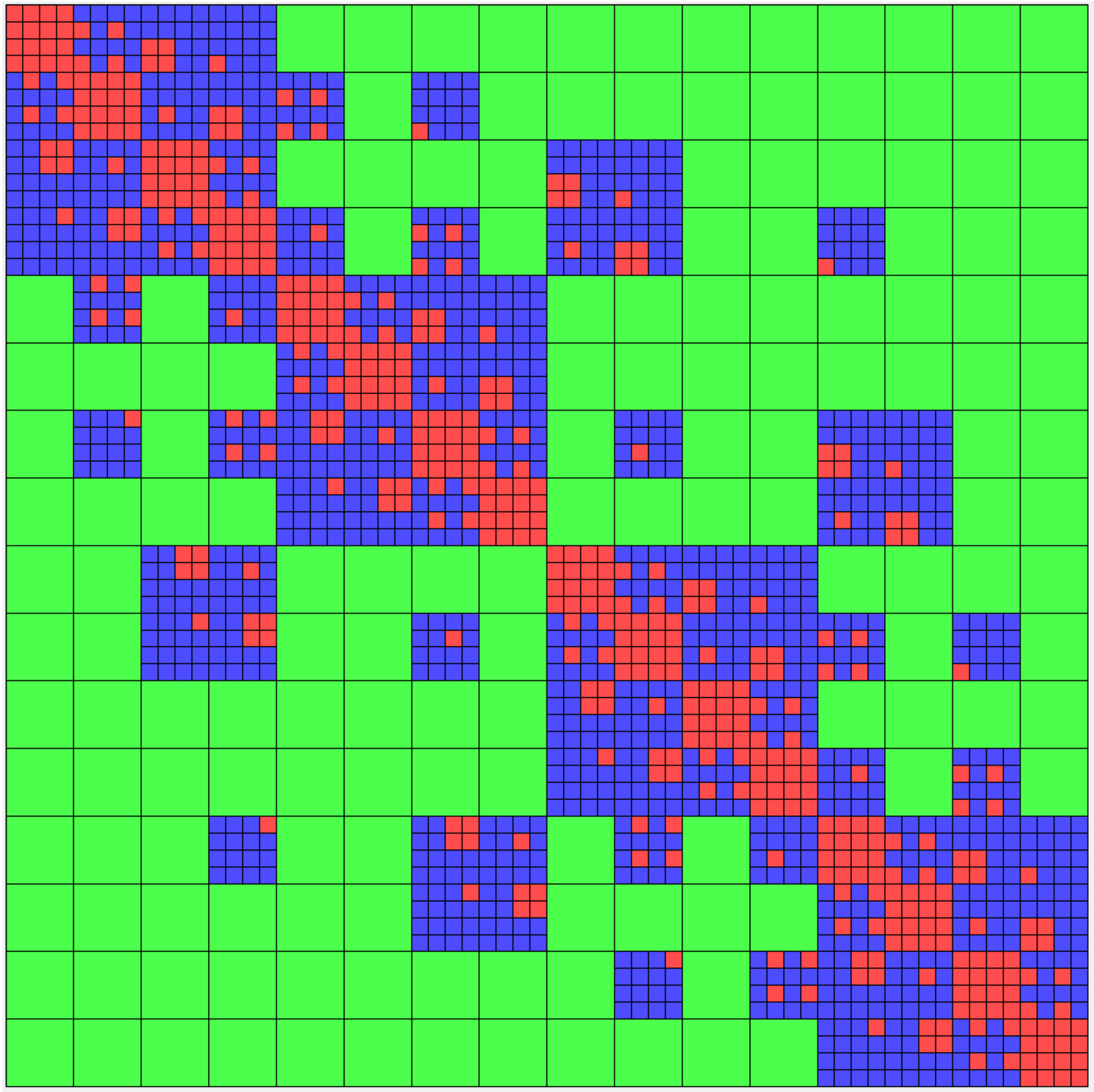
Low rank on level 2. Low rank on level 3. Low rank on level 4. Dense blocks. Gaussian blocks.

Say we know the green and yellow blocks, and seek to compress the blue blocks.

Key fact: At each level, we now need 6 block columns in **R**, rather than just two.

Complication 2: Dimension two (and strong admissibility)

For a problem in two dimensions, the tessellation of the matrix gets more complicated:



Low rank on level 2. *Low rank on level 3.* *Dense blocks.*

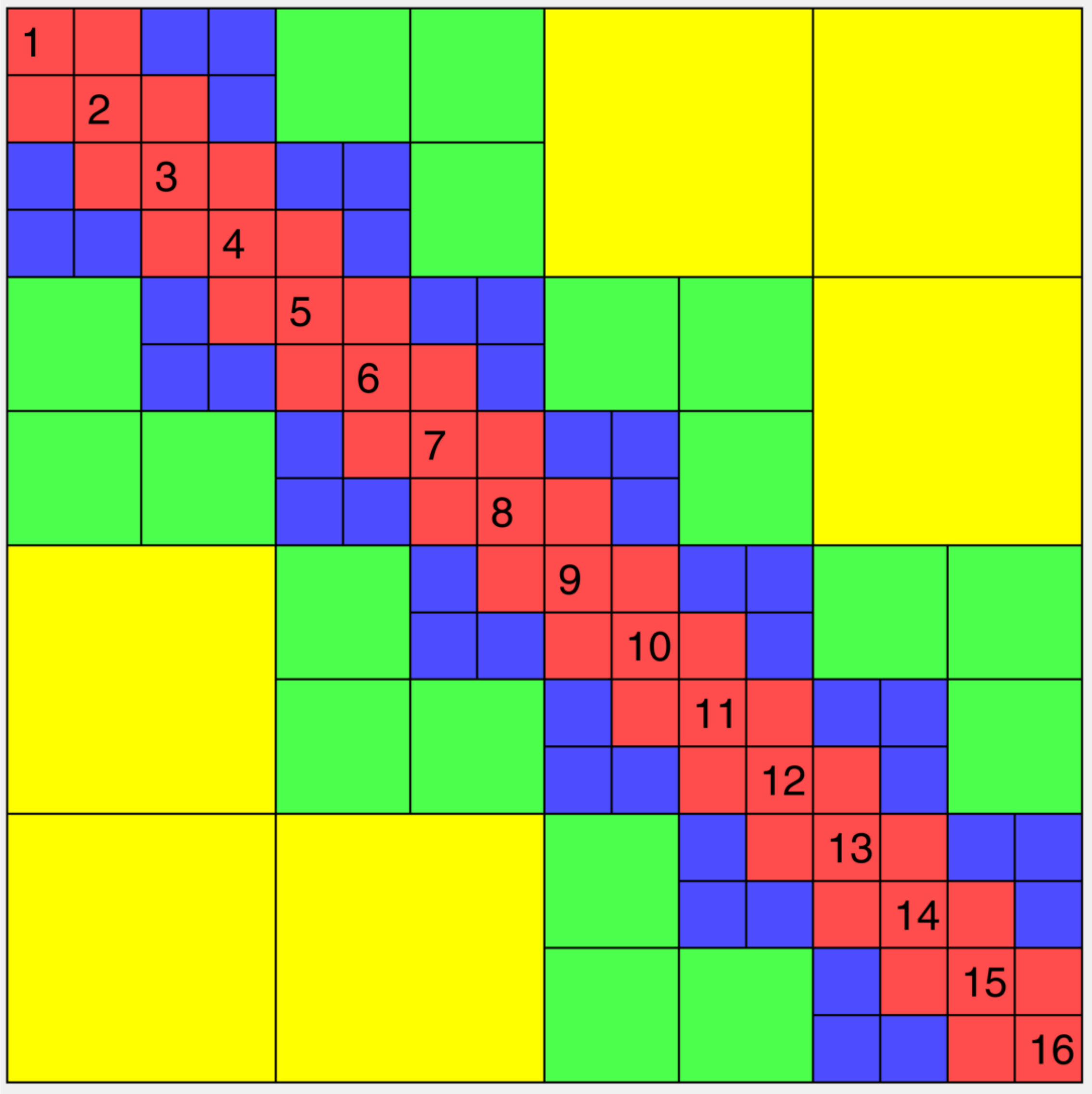
Lin, Li, Ying (2011): Build **R** that contains 64 blocks of k columns each.

If extended to dimension d, the method leads to 8^d block columns.

Question: Can these numbers be improved?

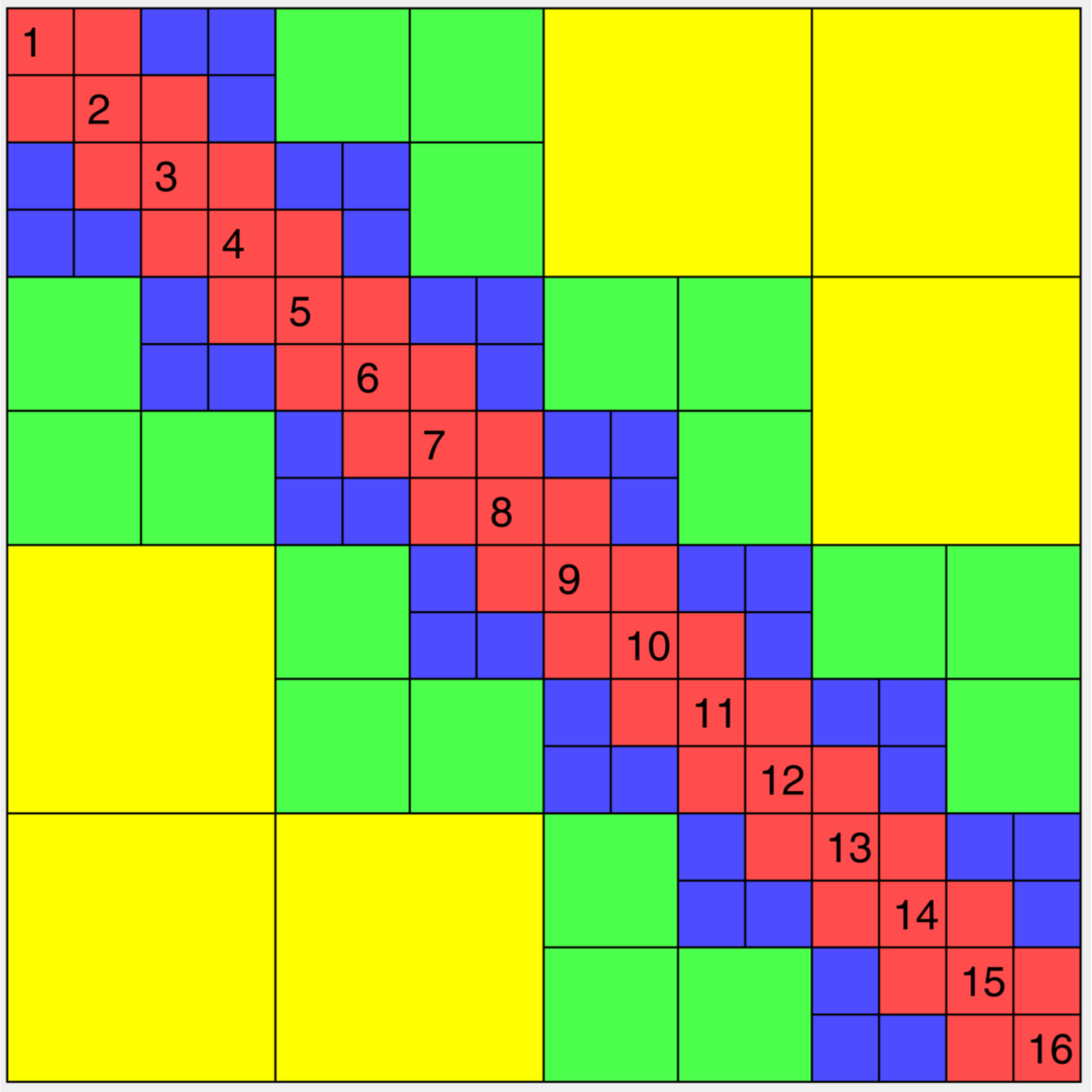
A graph coloring problem

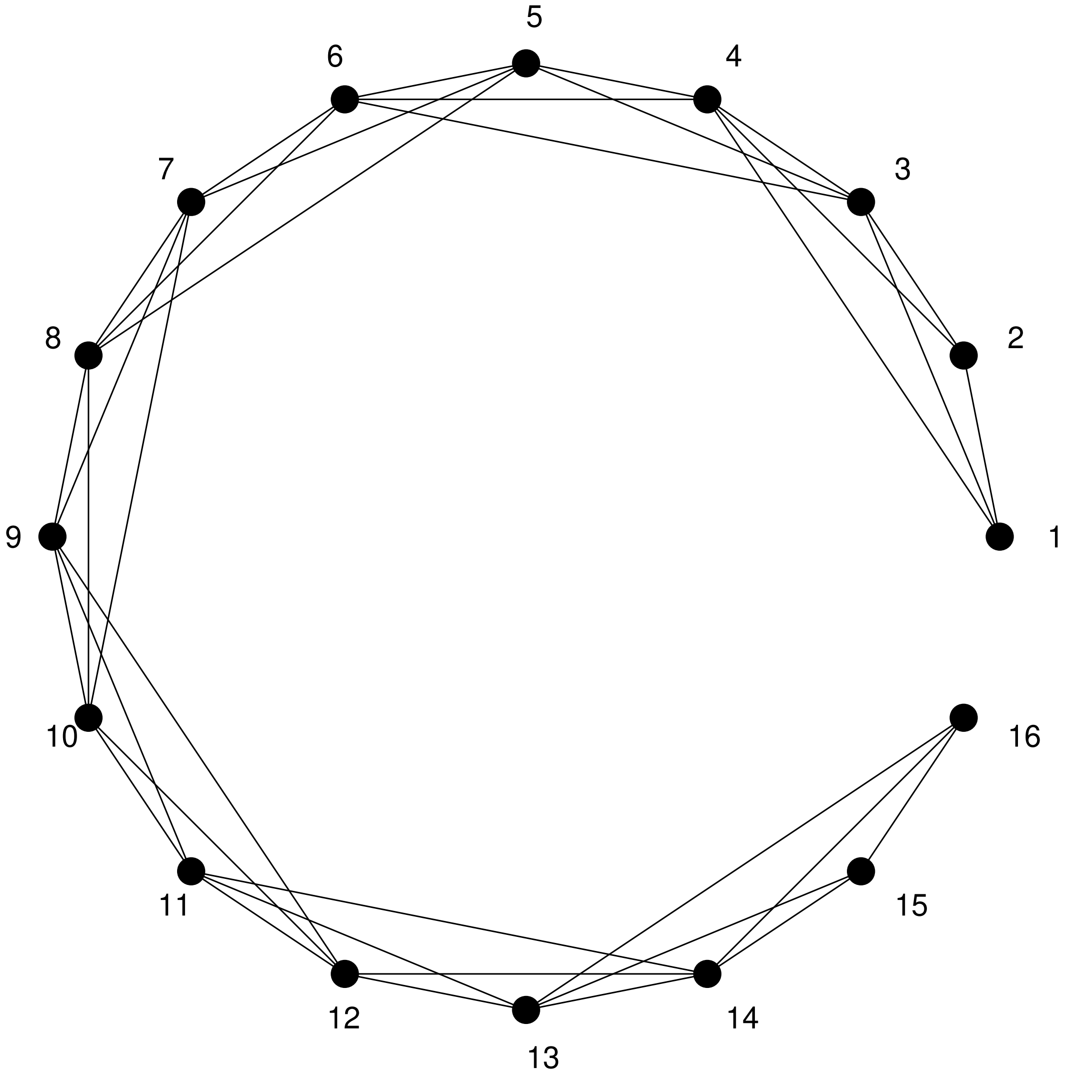
The task of finding a close to optimal matrix **R** can be viewed as finding a solution to a particular graph coloring problem. Let us revisit the case of strong admissibility in one dimension:



A graph coloring problem

The task of finding a close to optimal matrix **R** can be viewed as finding a solution to a particular graph coloring problem. Let us revisit the case of strong admissibility in one dimension:

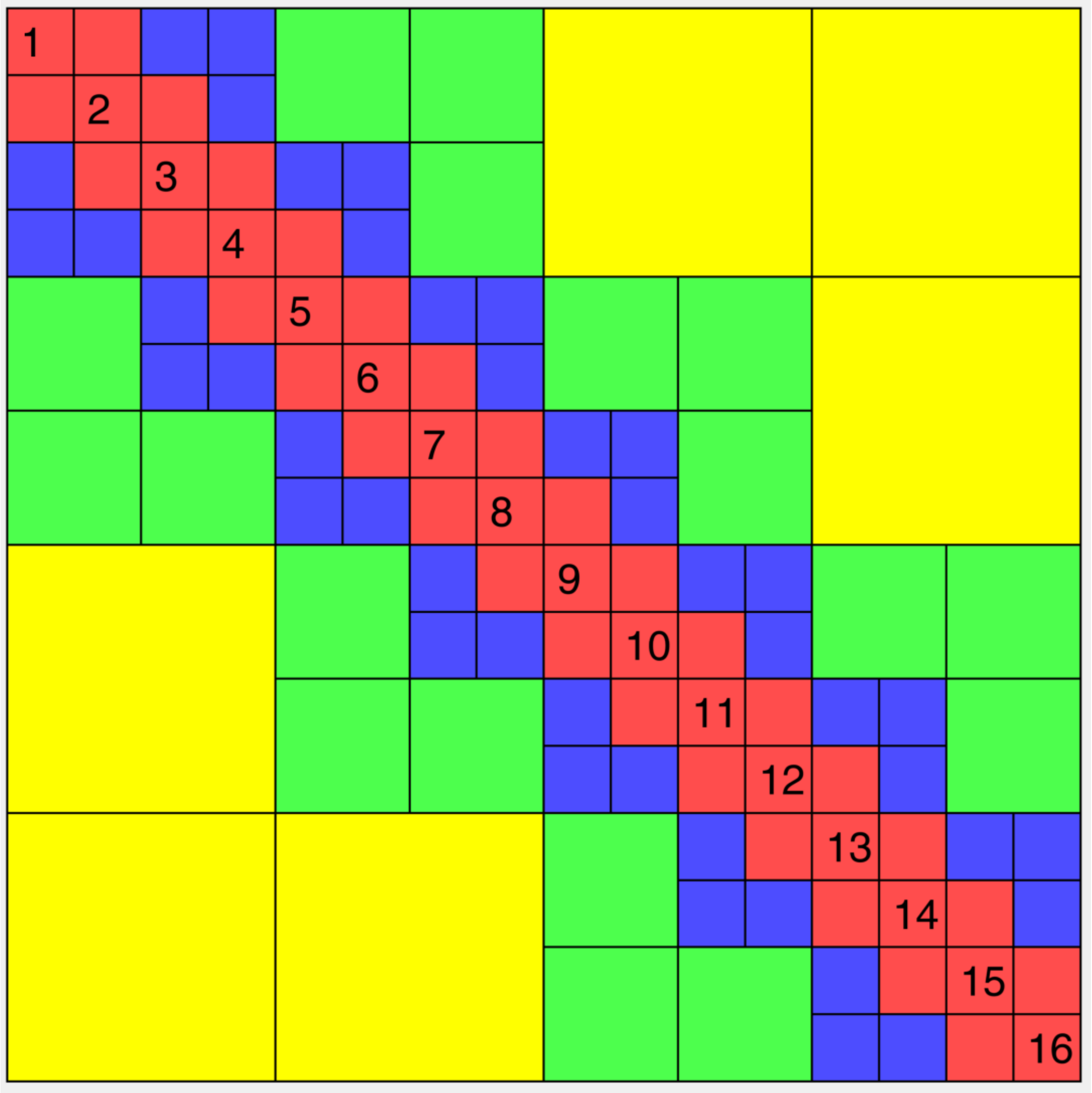


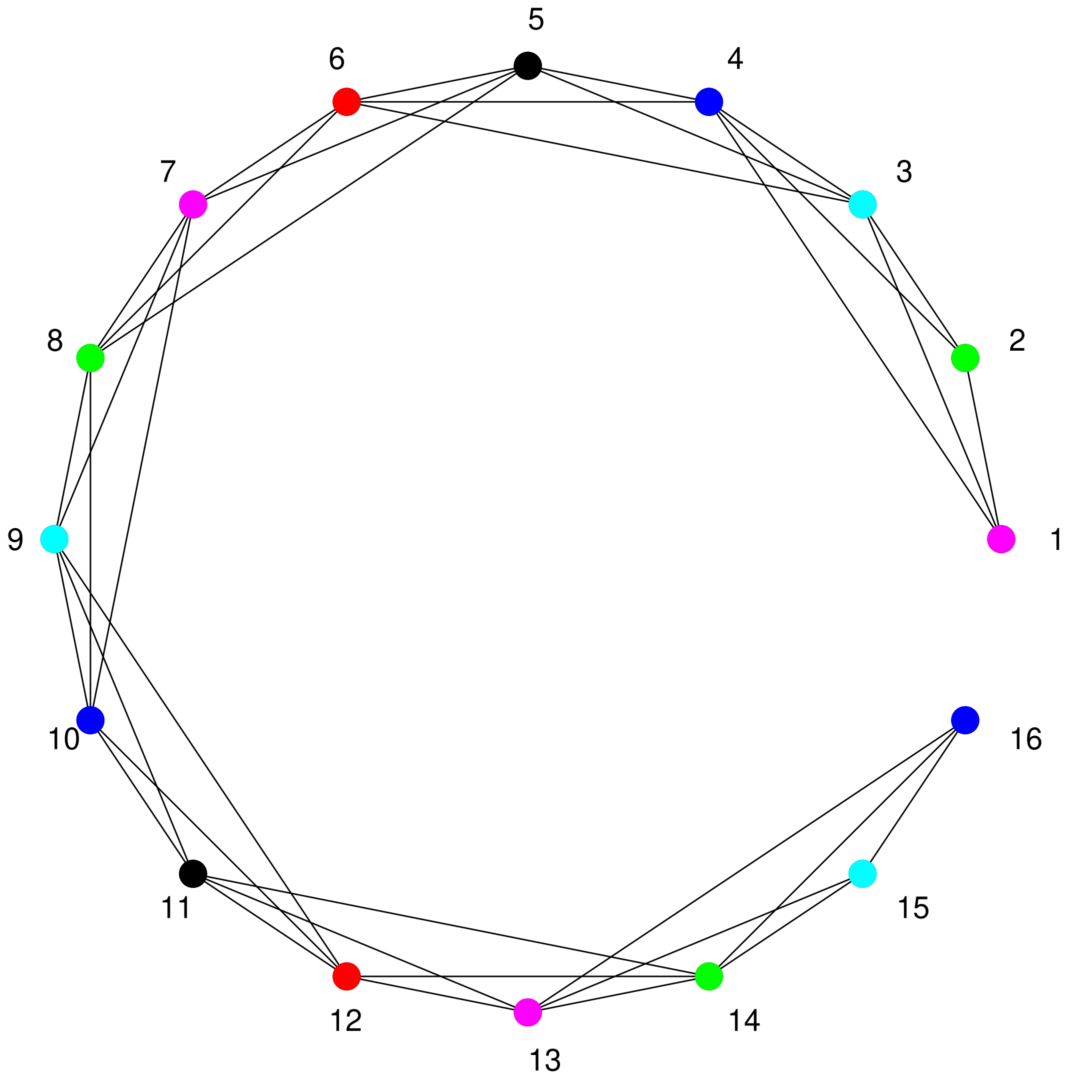


Each line represents a "conflict".

A graph coloring problem

The task of finding a close to optimal matrix **R** can be viewed as finding a solution to a particular graph coloring problem. Let us revisit the case of strong admissibility in one dimension:

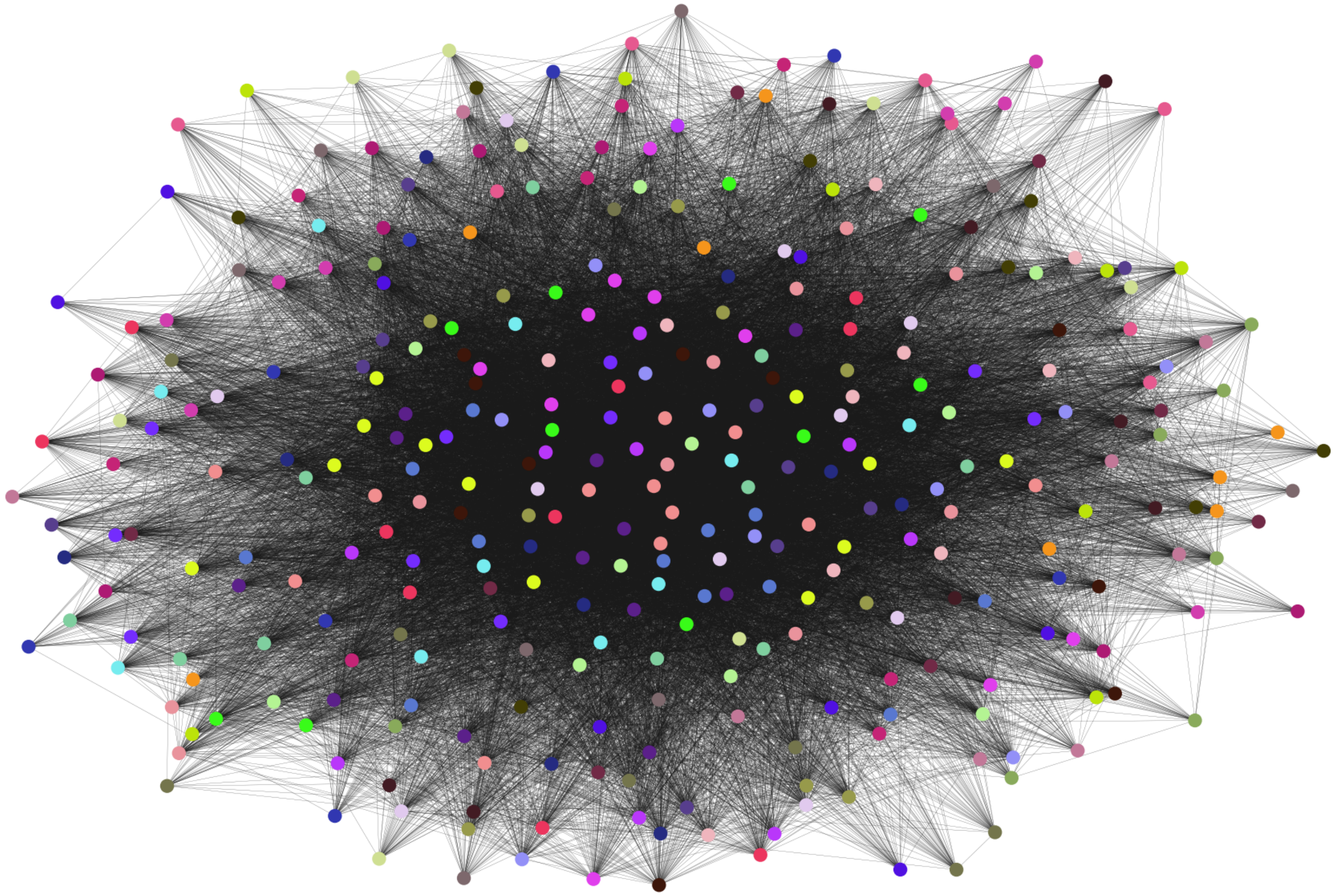




Each line represents a "conflict".

The task is to color the nodes in such a way that no conflicting nodes share a color. We solved this task using the "DSatur" graph coloring algorithm of D. Brélaz (1979).

Two dimensions with strong admissibility



Empirical discovery: 36 blocks of columns were sufficient.

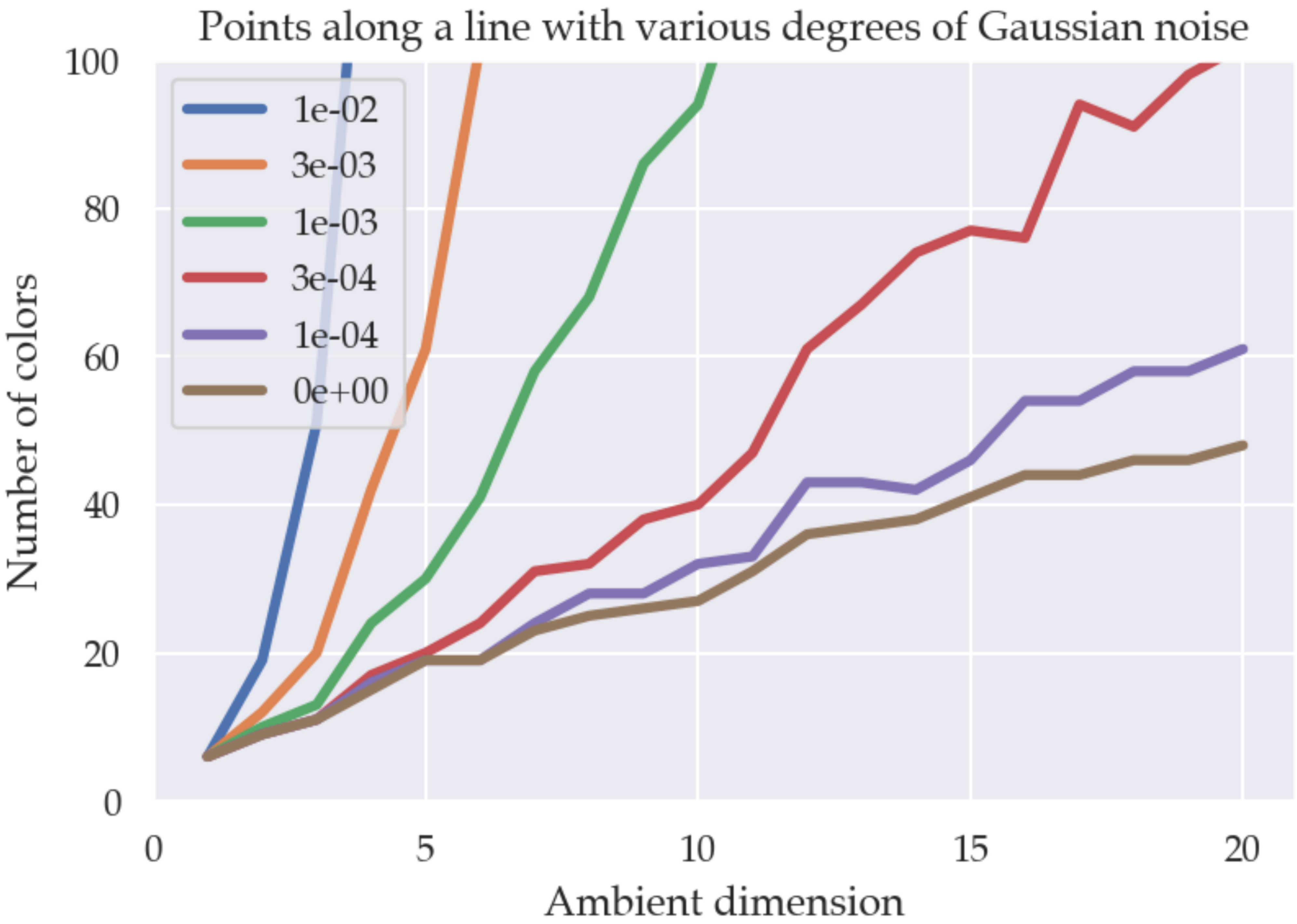
Inspecting the computed solutions, we found an explicit way to build **R**.

New theoretical finding: Need only 6^d blocks of columns, instead of 8^d, in worst case.

More importantly: The method can be run for any given set of points, to find an **R** that is optimized for that particular case. *Typically far fewer than 6^d colors needed!*

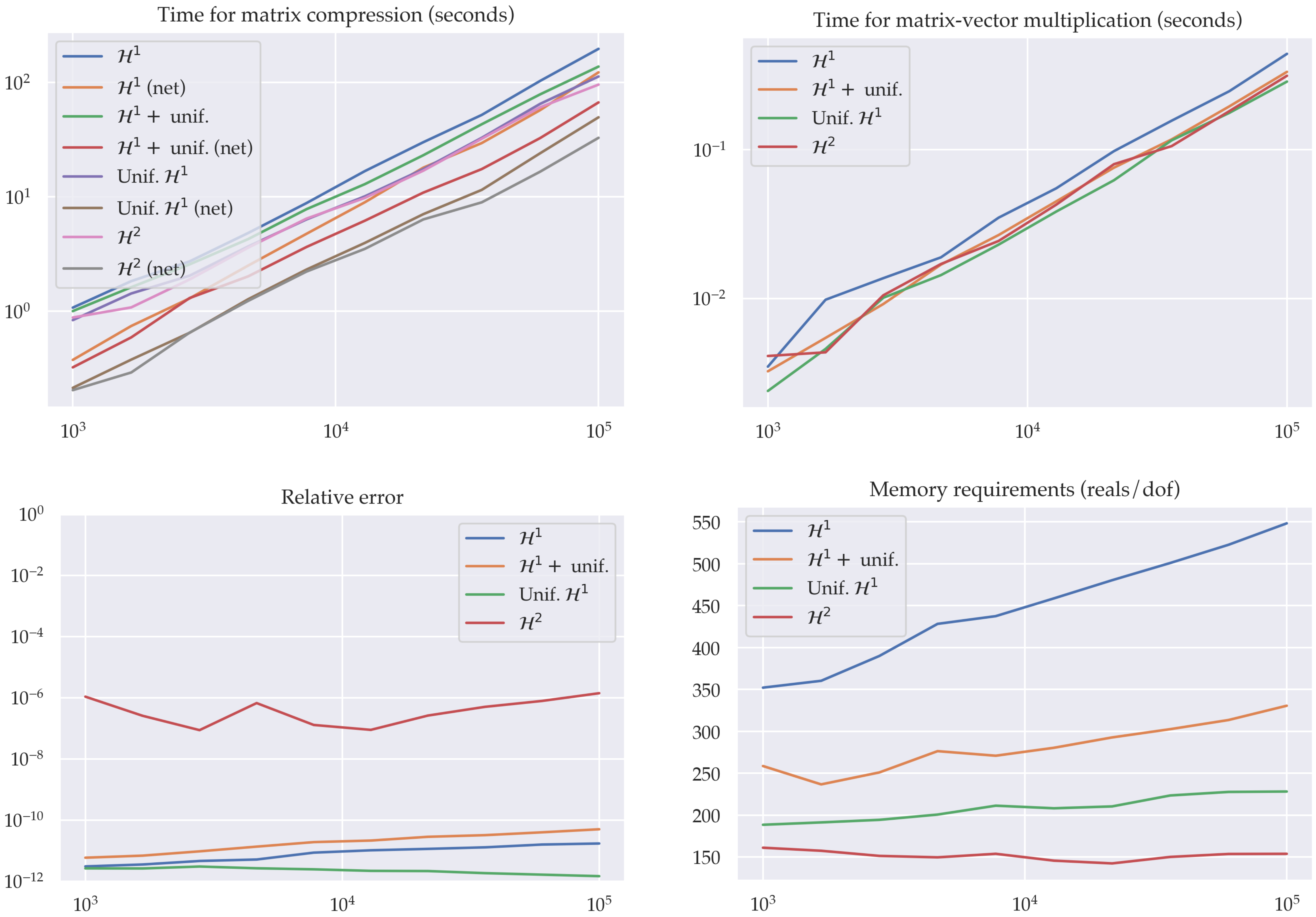
Example: Find the “intrinsic dimension”

The geometry is a line in $\mathbb{R}^d$ with Gaussian noise added.



Example: Computing the Neumann-to-Dirichlet operator for a 2D domain

Idea is to multiply together two boundary integral operators: $T = S\left(\frac{1}{2}I + D^*\right)^{-1}$.



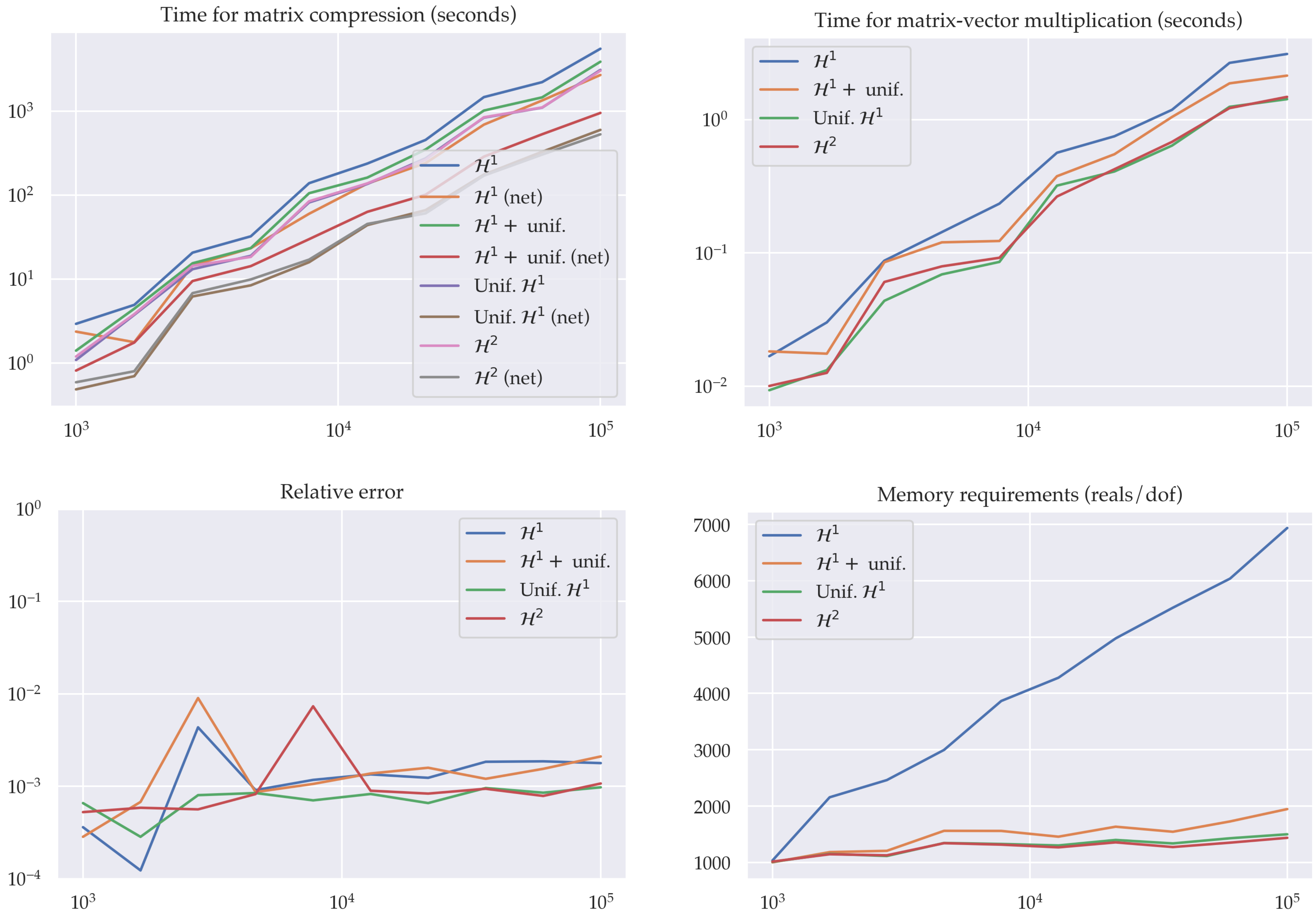
Time for matrix compression (seconds)

Time for matrix-vector multiplication (seconds)

Relative error

Memory requirements (reals/dof)

Example: FMM on two dimensional surface in 3D (Laplace problem)



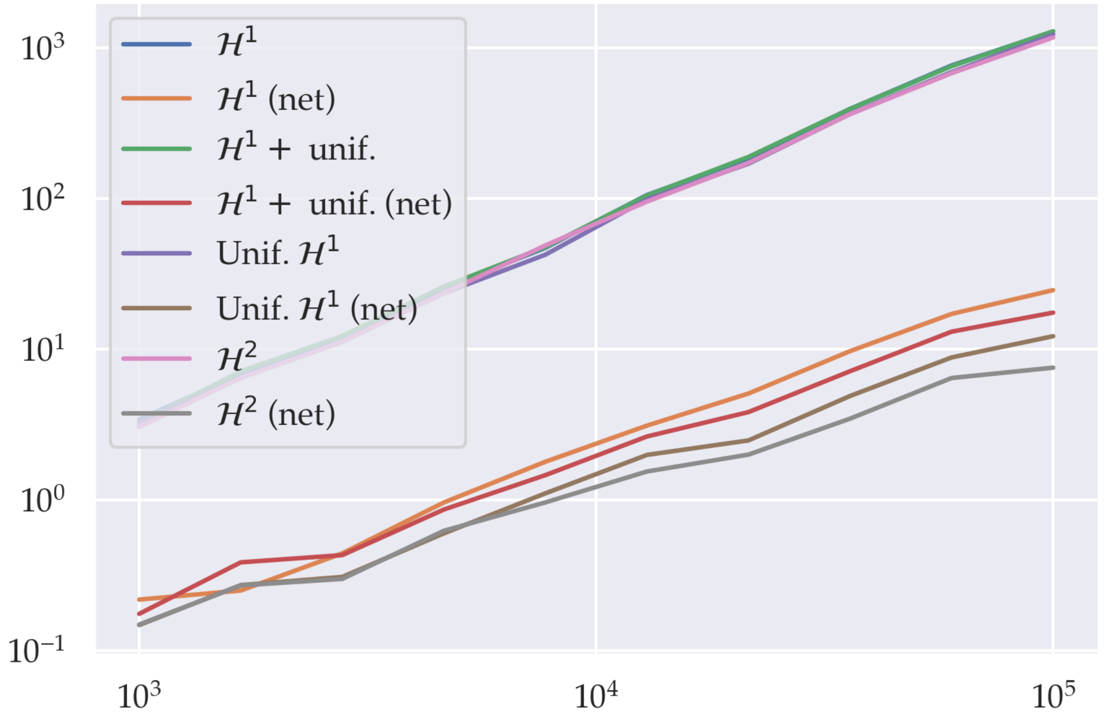
Note: This would have required 512×rank matvecs per level in the original scheme.

Example: Frontal matrix in nested dissection sparse direct solvers

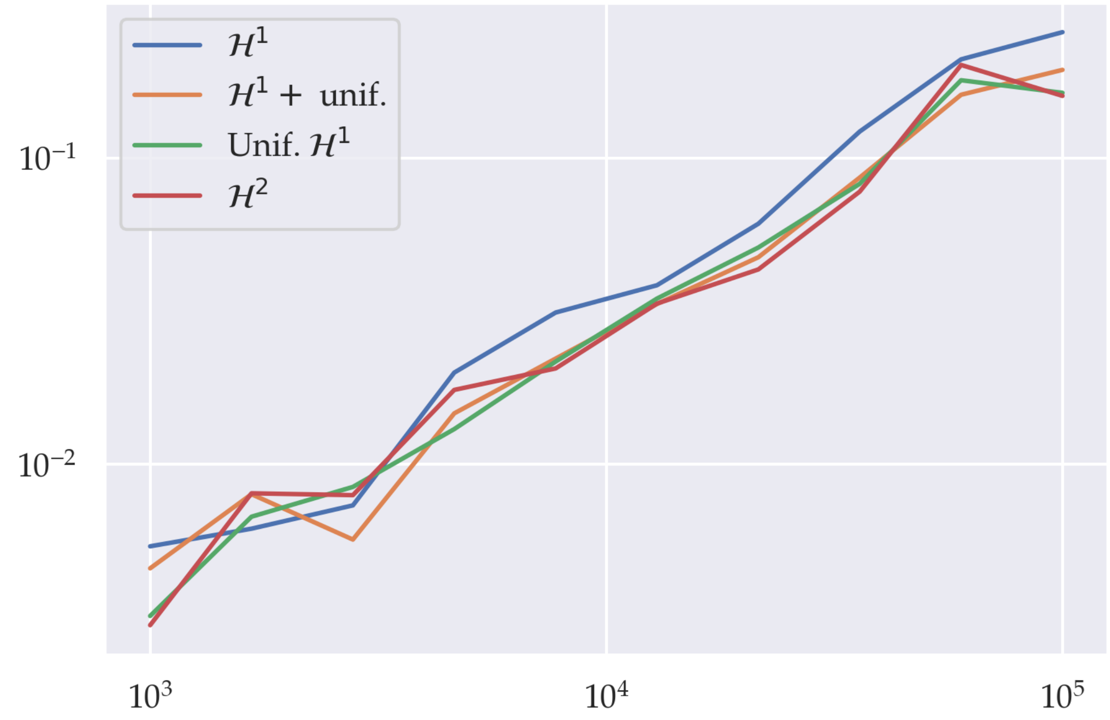
Let $\mathbf{B}$ be a finite difference matrix arising from discretizing an elliptic PDE.

Then $\mathbf{A}$ is a Schur complement: $\mathbf{A} = \mathbf{B}_{33} - \mathbf{B}_{31}\mathbf{B}_{11}^{-1}\mathbf{B}_{13} - \mathbf{B}_{32}\mathbf{B}_{22}^{-1}\mathbf{B}_{23}$

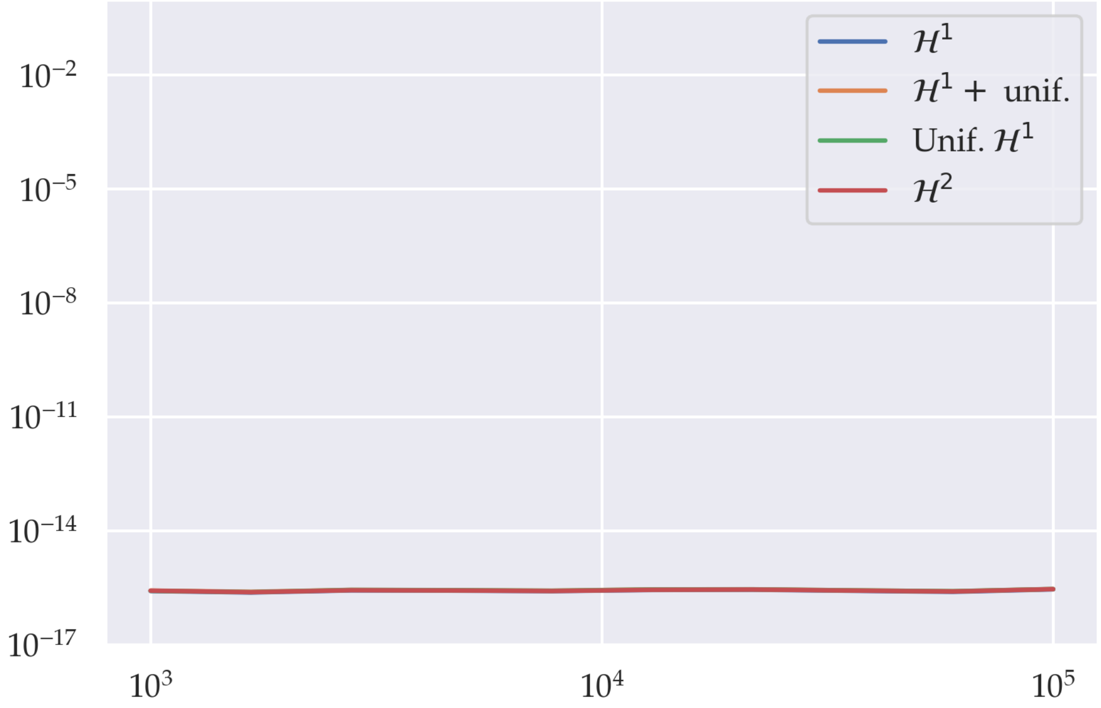
Time for matrix compression (seconds)



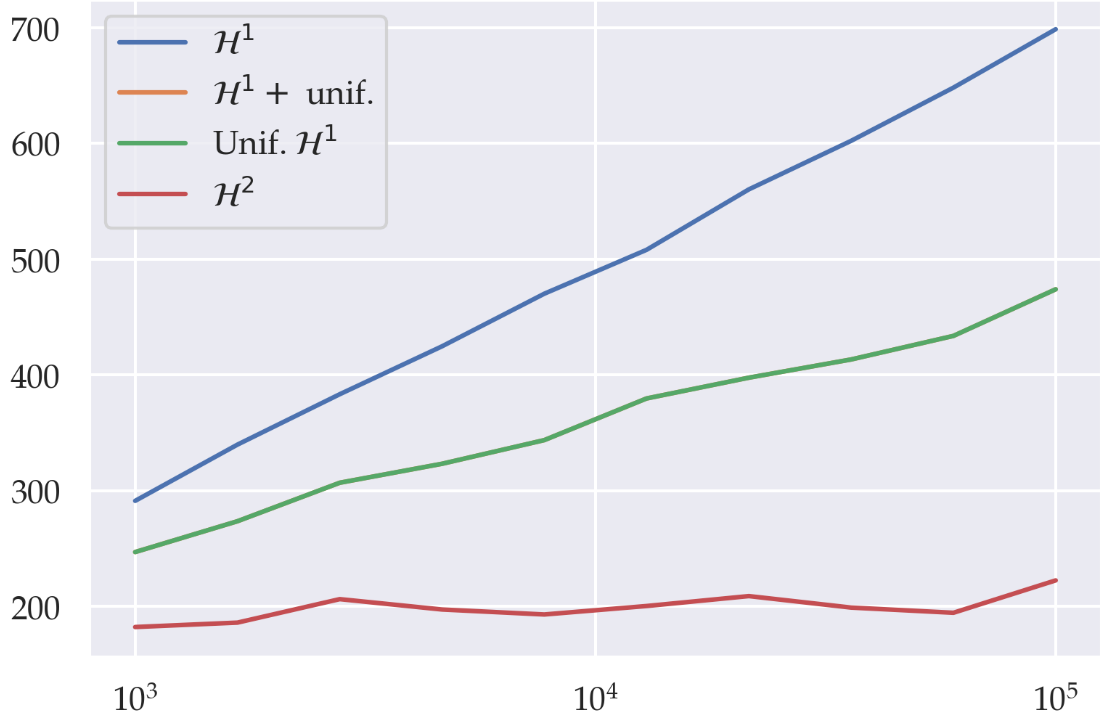
Time for matrix-vector multiplication (seconds)



Relative error



Memory requirements (reals/dof)



Summary

This presentation:

- Global operators form a central role in scientific computing.
- Linear complexity algorithms exist for many tasks:
  - Potential evaluation via the FMM. *Well established.*
  - Inversion, exponentiation, spectral decompositions, . . . *In progress.*
  - Randomized algorithms for compression. . . . *In progress.*
- The work presented relies on linearity in a fundamental way.

Possible connections to Machine Learning:

- Could the methods presented be helpful in understanding how to represent global operators in ML models? The multiscale representation is designed to exploit that not much information gets transported between disjoint subdomains.
- Could ML techniques be combined with the methods we have to tackle certain non-linear problems? Inverse problems perhaps?
- Could these methods be used to provide “reduced models” for linear sub components in a larger multiphysics model?