

2024 IABEM Workshop at SUSTech, Shenzhen, China

Fast Direct Solvers for BIE/BEM

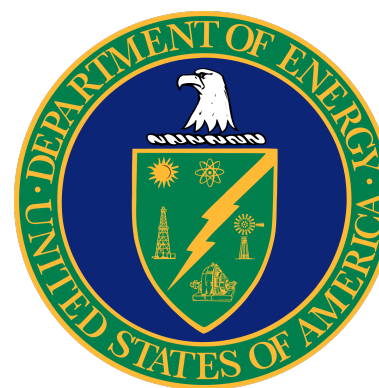
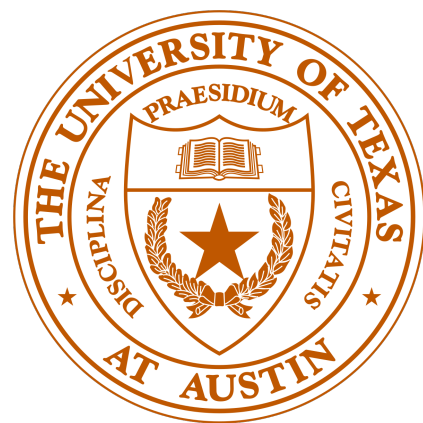
(and volume integral equations and some PDEs ...)

Gunnar Martinsson

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

University of Texas at Austin

Research support by:



Problem addressed: How do you efficiently compute approximate solutions to linear boundary value problems (BVPs) of the form

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

where Ω is a domain in \mathbb{R}^2 or \mathbb{R}^3 with boundary Γ , and where A is an elliptic differential operator (constant coefficient, or not). Examples include:

- The Laplace equation.
- 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).

In this lecture, focus will be on *integral equation formulations*:

$$c q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = f(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

Problem addressed: How do you efficiently compute approximate solutions to linear boundary value problems (BVPs) of the form

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

where Ω is a domain in \mathbb{R}^2 or \mathbb{R}^3 with boundary Γ , and where A is an elliptic differential operator (constant coefficient, or not). Examples include:

- The Laplace equation.
- 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).

Connection to Prof. Pan lecture: The solution operator to (BVP) takes the form

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

where G and F are the Green's functions. Our objective is in a sense to numerically build an approximation to the solution operator (SLN). Observe that (SLN) is mathematically benign: *smoothing operator, bounded, sometimes compact, ...*

Review: Integral equation formulation of BVPs

Most integral equation formulations rely on the concept of a *fundamental solution*.

To illustrate, suppose first that we seek to solve a **free space** Helmholtz equation

$$-\Delta u(\mathbf{x}) - \kappa^2 u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^3$$

where f is some given source function (typically with compact support), and κ is the wave-number. The PDE is coupled with a **radiation condition** at infinity

$$\lim_{|\mathbf{x}| \rightarrow \infty} |\mathbf{x}| \left(\partial_{|\mathbf{x}|} u(\mathbf{x}) - i\kappa u(\mathbf{x}) \right) = 0$$

to ensure well-posedness.

Review: Integral equation formulation of BVPs

Most integral equation formulations rely on the concept of a *fundamental solution*.

To illustrate, suppose first that we seek to solve a *free space* Helmholtz equation

$$-\Delta u(\mathbf{x}) - \kappa^2 u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^3$$

where f is some given source function (typically with compact support), and κ is the wave-number. The PDE is coupled with a *radiation condition* at infinity

$$\lim_{|\mathbf{x}| \rightarrow \infty} |\mathbf{x}| \left(\partial_{|\mathbf{x}|} u(\mathbf{x}) - i\kappa u(\mathbf{x}) \right) = 0$$

to ensure well-posedness.

In this case, we can write down the solution explicitly as

$$u(\mathbf{x}) = [\phi_\kappa * f](\mathbf{x}) = \int_{\mathbb{R}^3} \phi_\kappa(\mathbf{x} - \mathbf{y}) f(\mathbf{y}) d\mathbf{y}$$

where ϕ_κ is the *free space fundamental solution* to the Helmholtz equation

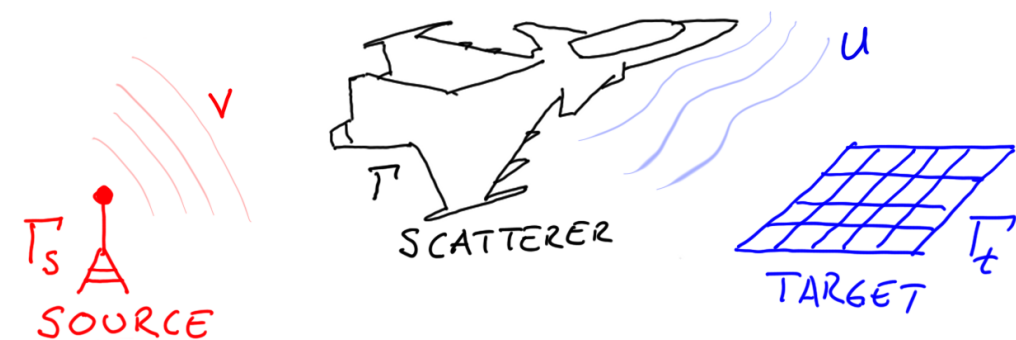
$$\phi_\kappa(\mathbf{x}) = \frac{e^{i\kappa|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|}.$$

This function satisfies the radiation condition at infinity, as well as

$$\Delta_{\mathbf{x}} \phi_\kappa(\mathbf{x} - \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y}).$$

Review: Integral equation formulation of BVPs

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:



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

The BVP (3) has an alternative mathematical formulation in the BIE

$$(4) \quad -\pi i \sigma(\mathbf{x}) + \int_{\partial\Omega} \left(\left(\partial_{\mathbf{n}(\mathbf{y})} + i\kappa \right) \frac{e^{i\kappa|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|} \right) \sigma(\mathbf{y}) d\mathbf{y} = f(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega.$$

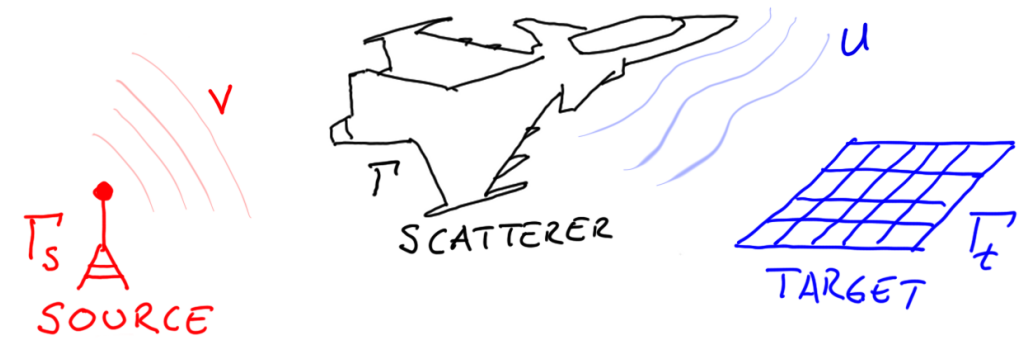
The integral equation (4) has several advantages over the PDE (3), including:

- The domain of computation $\partial\Omega$ is finite.
- The domain of computation $\partial\Omega$ is 2D, while $\mathbb{R}^3 \setminus \bar{\Omega}$ is 3D.
- Equation (4) is inherently well-conditioned (as a “2nd kind Fredholm equation”).

A challenge of integral equations is that they lead to *dense coefficient matrices*.

Review: Integral equation formulation of BVPs

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:



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

The BVP (3) has an alternative mathematical formulation in the BIE

$$(4) \quad -\pi i \sigma(\mathbf{x}) + \int_{\partial\Omega} \left(\left(\partial_{\mathbf{n}(\mathbf{y})} + i\kappa \right) \frac{e^{i\kappa|\mathbf{x}-\mathbf{y}|}}{|\mathbf{x}-\mathbf{y}|} \right) \sigma(\mathbf{y}) d\mathbf{y} = f(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega.$$

The integral equation (4) has several advantages over the PDE (3), including:

- The domain of computation $\partial\Omega$ is finite.
- The domain of computation $\partial\Omega$ is 2D, while $\mathbb{R}^3 \setminus \bar{\Omega}$ is 3D.
- Equation (4) is inherently well-conditioned (as a “2nd kind Fredholm equation”).

For constant coefficient problems, we can often reformulate a 3D sparse problem as a 2D dense one.

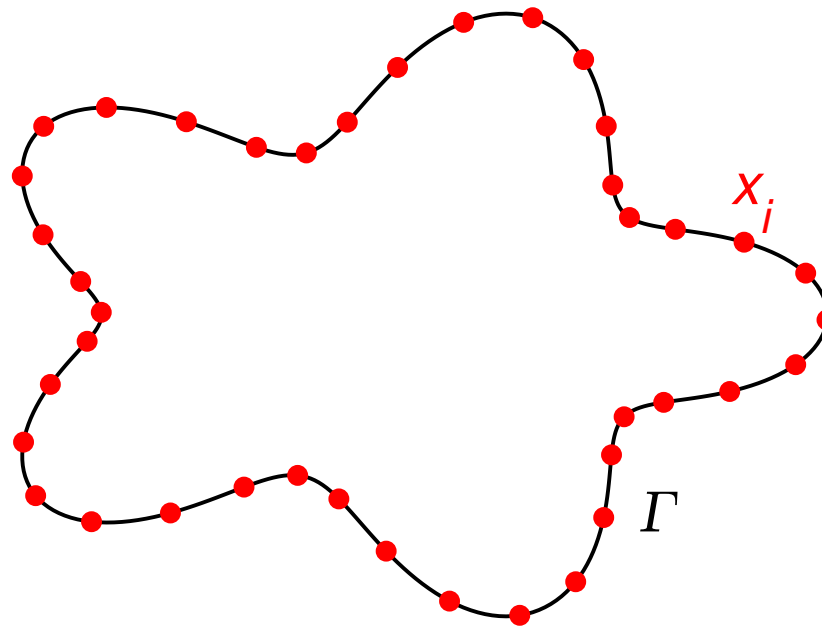
Review: Nyström discretization of BIE: Consider an integral equation of the form

$$(BIE) \quad \frac{1}{2}q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

We discretize (BIE) using the Nyström method based on a quadrature rule

$$\int_{\Gamma} \varphi(\mathbf{y}) d\mathbf{y} \approx \sum_{i=1}^N w_i \varphi(\mathbf{x}_i)$$

designed for *smooth* functions φ .



Example: For a smooth contour Γ , you can use the Trapezoidal rule to build a quadrature rule. If the contour is parameterized via

$$\mathbf{G} : [0, L] \rightarrow \mathbb{R}^2 : t \mapsto \mathbf{G}(t),$$

then you can use

$$\mathbf{x}_i = \mathbf{G}(ih), \quad w_i = h|\mathbf{G}'(ih)|$$

where $h = L/N$ is the grid spacing in parameter space.

Review: Nyström discretization of BIE: Consider an integral equation of the form

$$(BIE) \quad \frac{1}{2}q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

We discretize (BIE) using the Nyström method based on a quadrature rule

$$\int_{\Gamma} \varphi(\mathbf{y}) d\mathbf{y} \approx \sum_{i=1}^N w_i \varphi(\mathbf{x}_i)$$

designed for *smooth* functions φ .

Review: Nyström discretization of BIE: Consider an integral equation of the form

$$(BIE) \quad \frac{1}{2}q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

We discretize (BIE) using the Nyström method based on a quadrature rule

$$\int_{\Gamma} \varphi(\mathbf{y}) d\mathbf{y} \approx \sum_{i=1}^N w_i \varphi(\mathbf{x}_i)$$

designed for *smooth* functions φ . Now collocate (BIE) at the quadrature nodes:

$$\frac{1}{2}q(\mathbf{x}_i) + \int_{\Gamma} k(\mathbf{x}_i, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}_i), \quad i = 1, 2, 3, \dots, N.$$

Then construct an $N \times N$ matrix \mathbf{A} such that

$$(Q) \quad \sum_{j=1}^N \mathbf{A}(i, j) q(\mathbf{x}_j) \approx \int_{\Gamma} k(\mathbf{x}_i, \mathbf{y}) q(\mathbf{y}) d\mathbf{y}, \quad i = 1, 2, 3, \dots, N.$$

This can often be done so that the vast majority of elements of \mathbf{A} take the form

$$\mathbf{A}(i, j) = k(\mathbf{x}_i, \mathbf{x}_j) w_j.$$

The discretized version of (BIE) is then

$$\mathbf{A} \mathbf{q} = \mathbf{v},$$

which relates the given vector \mathbf{v} with entries $\mathbf{v}(i) = v(\mathbf{x}_i)$ to the sought vector \mathbf{q} whose elements should (hopefully!) satisfy $\mathbf{q}_i \approx q(\mathbf{x}_i)$.

Review: Nyström discretization of BIE: Consider an integral equation of the form

$$(BIE) \quad \frac{1}{2}q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

We discretize (BIE) using the Nyström method based on a quadrature rule

$$\int_{\Gamma} \varphi(\mathbf{y}) d\mathbf{y} \approx \sum_{i=1}^N w_i \varphi(\mathbf{x}_i)$$

designed for *smooth* functions φ . Now collocate (BIE) at the quadrature nodes:

$$\frac{1}{2}q(\mathbf{x}_i) + \int_{\Gamma} k(\mathbf{x}_i, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}_i), \quad i = 1, 2, 3, \dots, N.$$

Then construct an $N \times N$ matrix \mathbf{A} such that

$$(Q) \quad \sum_{j=1}^N \mathbf{A}(i, j) q(\mathbf{x}_j) \approx \int_{\Gamma} k(\mathbf{x}_i, \mathbf{y}) q(\mathbf{y}) d\mathbf{y}, \quad i = 1, 2, 3, \dots, N.$$

This can often be done so that the vast majority of elements of \mathbf{A} take the form

$$\mathbf{A}(i, j) = k(\mathbf{x}_i, \mathbf{x}_j) w_j.$$

Note: Constructing matrix elements that ensure that (Q) holds is delicate, since $k(\mathbf{x}, \mathbf{y})$ is almost always singular at the diagonal. Many papers written on this subject! Luckily, this complication is localized to a small number of entries of the matrix.

Review: Nyström discretization of BIE: Consider an integral equation of the form

$$(BIE) \quad \frac{1}{2}q(\mathbf{x}) + \int_{\Gamma} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

We discretize (BIE) using the Nyström method based on a quadrature rule

$$\int_{\Gamma} \varphi(\mathbf{y}) d\mathbf{y} \approx \sum_{i=1}^N w_i \varphi(\mathbf{x}_i)$$

designed for *smooth* functions φ . Now collocate (BIE) at the quadrature nodes:

$$\frac{1}{2}q(\mathbf{x}_i) + \int_{\Gamma} k(\mathbf{x}_i, \mathbf{y}) q(\mathbf{y}) d\mathbf{y} = v(\mathbf{x}_i), \quad i = 1, 2, 3, \dots, N.$$

Then construct an $N \times N$ matrix \mathbf{A} such that

$$(Q) \quad \sum_{j=1}^N \mathbf{A}(i, j) q(\mathbf{x}_j) \approx \int_{\Gamma} k(\mathbf{x}_i, \mathbf{y}) q(\mathbf{y}) d\mathbf{y}, \quad i = 1, 2, 3, \dots, N.$$

This can often be done so that the vast majority of elements of \mathbf{A} take the form

$$\mathbf{A}(i, j) = k(\mathbf{x}_i, \mathbf{x}_j) w_j.$$

Note: When *boundary element methods* are used, the derivation is different. But almost everything in this lecture works for both BEM and Nyström. The key property is that *most* elements of \mathbf{A} takes the form $\mathbf{A}(i, j) = v_i k(\mathbf{x}_i, \mathbf{x}_j) w_j$, which holds in both cases.

Discretization of linear Boundary Value Problems

↙

Direct discretization of the differential operator via Finite Elements, Finite Differences, spectral composite methods, ...

↓

$N \times N$ discrete linear system.
Very large, sparse, **ill-conditioned**.

↓

Solver methodologies:

- Iterative (e.g. multigrid). Often $O(N)$.
- Direct (e.g. nested dissection) . Typically between $O(N^{1.5})$ and $O(N^2)$.
-

↘

Conversion of the BVP to a Boundary Integral Equation (BIE).

↓

Discretization of (BIE) using Nyström, collocation, BEM,

↓

$N \times N$ discrete linear system.
Moderate size, dense,
(often) well-conditioned.

↓

Solver methodologies:

- Iterative solver accelerated by fast matrix-vector multiplier. Often $O(N)$.
- Direct solvers. $O(N^3)$
-

Discretization of linear Boundary Value Problems

↙

Direct discretization of the differential operator via Finite Elements, Finite Differences, spectral composite methods, ...

↓

$N \times N$ discrete linear system.
Very large, sparse, **ill-conditioned**.

↓

Solver methodologies:

- Iterative (e.g. multigrid). Often $O(N)$.
- Direct (e.g. nested dissection) . Typically between $O(N^{1.5})$ and $O(N^2)$.
- **Fast direct solvers. Often $O(N)$.**

↘

Conversion of the BVP to a Boundary Integral Equation (BIE).

↓

Discretization of (BIE) using Nyström, collocation, BEM,

↓

$N \times N$ discrete linear system.
Moderate size, dense,
(often) well-conditioned.

↓

Solver methodologies:

- Iterative solver accelerated by fast matrix-vector multiplier. Often $O(N)$.
- Direct solvers. $O(N^3)$
- **Fast direct solvers. Often $O(N)$.**

What does a “direct” solver mean in this context?

Basically, it is a solver that is not “iterative” ...

Given a computational tolerance ε , and a linear system

$$(7) \quad \mathbf{A} \mathbf{u} = \mathbf{b},$$

(where the system matrix \mathbf{A} is often defined implicitly), a *direct solver* constructs an operator \mathbf{S} such that

$$\|\mathbf{A}^{-1} - \mathbf{S}\| \leq \varepsilon.$$

Then an approximate solution to (7) is obtained by simply evaluating

$$\mathbf{u}_{\text{approx}} = \mathbf{S} \mathbf{b}.$$

The matrix \mathbf{S} is typically constructed in a *data-sparse format* (e.g. \mathcal{H} -matrix, HSS, etc) that allows the matrix-vector product $\mathbf{S} \mathbf{b}$ to be evaluated rapidly.

Variation: Find factors \mathbf{B} and \mathbf{C} such that $\|\mathbf{A} - \mathbf{B} \mathbf{C}\| \leq \varepsilon$, and linear solves involving the matrices \mathbf{B} and \mathbf{C} are fast. (LU-decomposition, Cholesky, etc.)

What does a “direct” solver mean in this context?

When using these algorithms, the process of solving (BVP) splits into two stages:

1. “**Factorization**” or “**build**” stage: Build a representation of the inverse operator.
2. “**Solve**” stage: Apply the computed inverse to given data f or (and) g .

Typical characteristics of methods of this type:

- Memory usage tends to be high. (Linear, but a large “pre-factor”.)
- Stage 1 tends to be slower than an iterative solve, *when convergence is fast*.
- Stage 2 is almost always VERY fast.

Fast Direct Solvers (FDS) are most competitive when:

- Getting iterative methods to converge rapidly is hard.
- When the cost of Stage 1 can be amortized over many solves.
 - scattering problems, time-stepping, optimization, ...

Advantages of direct solvers over iterative solvers:

1. Solving problems intractable to iterative methods (singular values do not “cluster”):
 - Scattering problems near resonant frequencies.
 - Ill-conditioning due to geometry (elongated domains, percolation, etc).
 - Ill-conditioning due to lazy handling of corners, cusps, etc.
 - Finite element and finite difference discretizations.

Scattering problems intractable to existing methods can (sometimes) be solved.

2. Applications that require a very large number of solves for a fixed operator:
 - Molecular dynamics.
 - Scattering problems.
 - Optimal design. (Local updates to the system matrix are cheap.)

A couple of orders of magnitude speed-up is often possible.

3. Direct solvers can be adapted to construct spectral decompositions:
 - Analysis of vibrating structures. Acoustics.
 - Buckling of mechanical structures.
 - Wave guides, bandgap materials, etc.

Work in progress . . .

Advantages of direct solvers over iterative solvers, continued:

Perhaps most important: **Engineering considerations.**

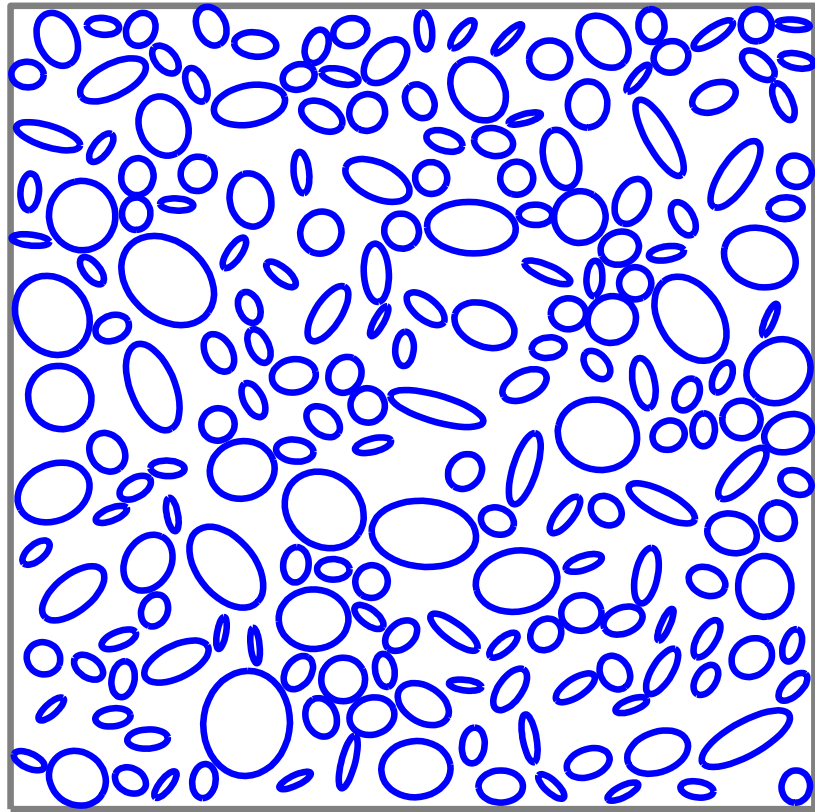
Direct methods tend to be more **robust** than iterative ones.

This makes them more suitable for “black-box” implementations.

Commercial software developers appear to avoid implementing iterative solvers whenever possible. (Sometimes for good reasons.)

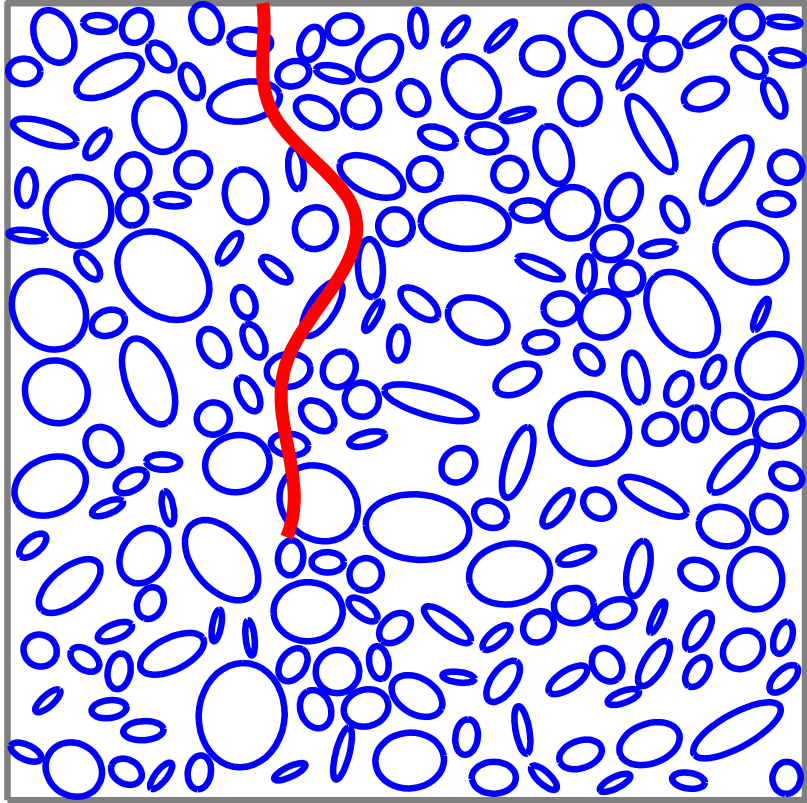
The effort to develop direct solvers aims to help in the development of general purpose software packages solving the basic linear boundary value problems of mathematical physics.

Advantages of direct solvers over iterative solvers, subtleties:



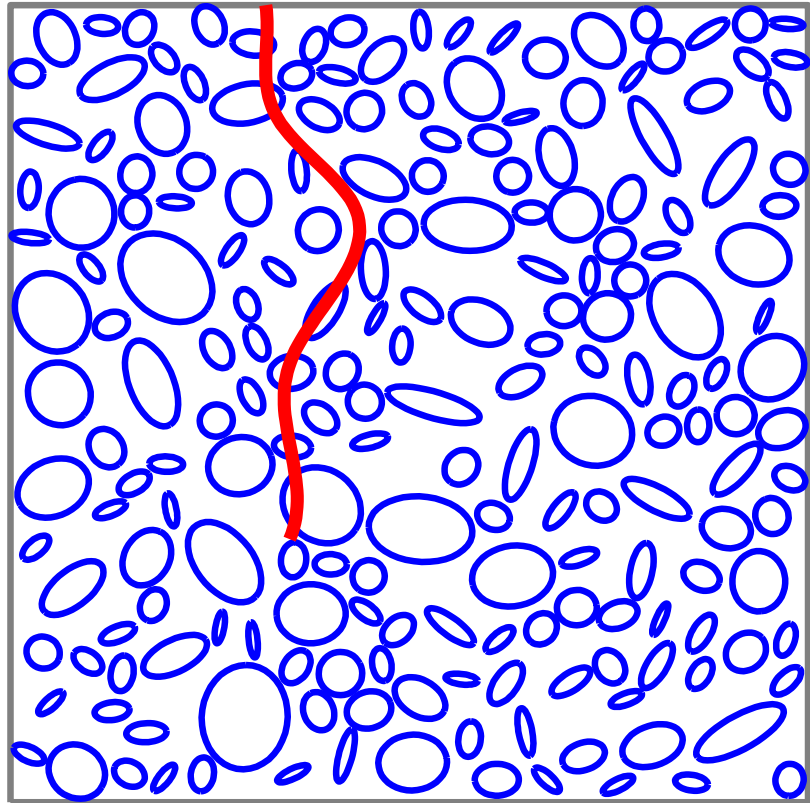
Numerical model reduction.

Advantages of direct solvers over iterative solvers, subtleties:

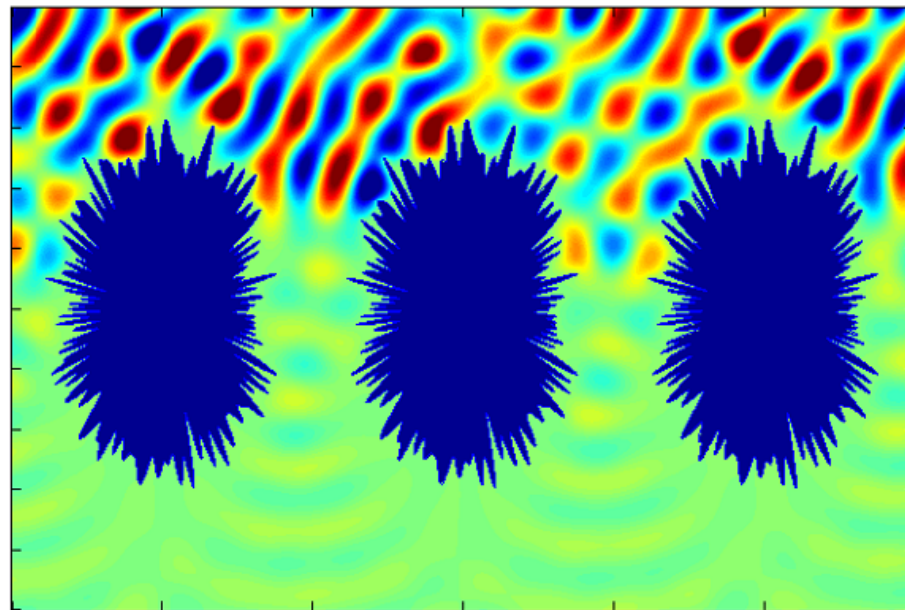


Numerical model reduction.

Advantages of direct solvers over iterative solvers, subtleties:

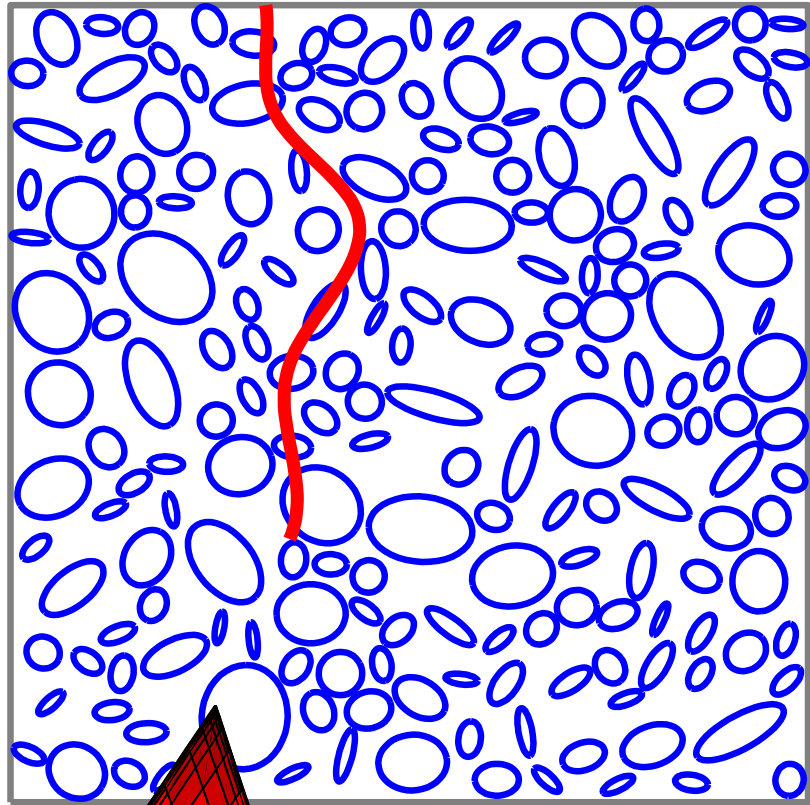


Numerical model reduction.

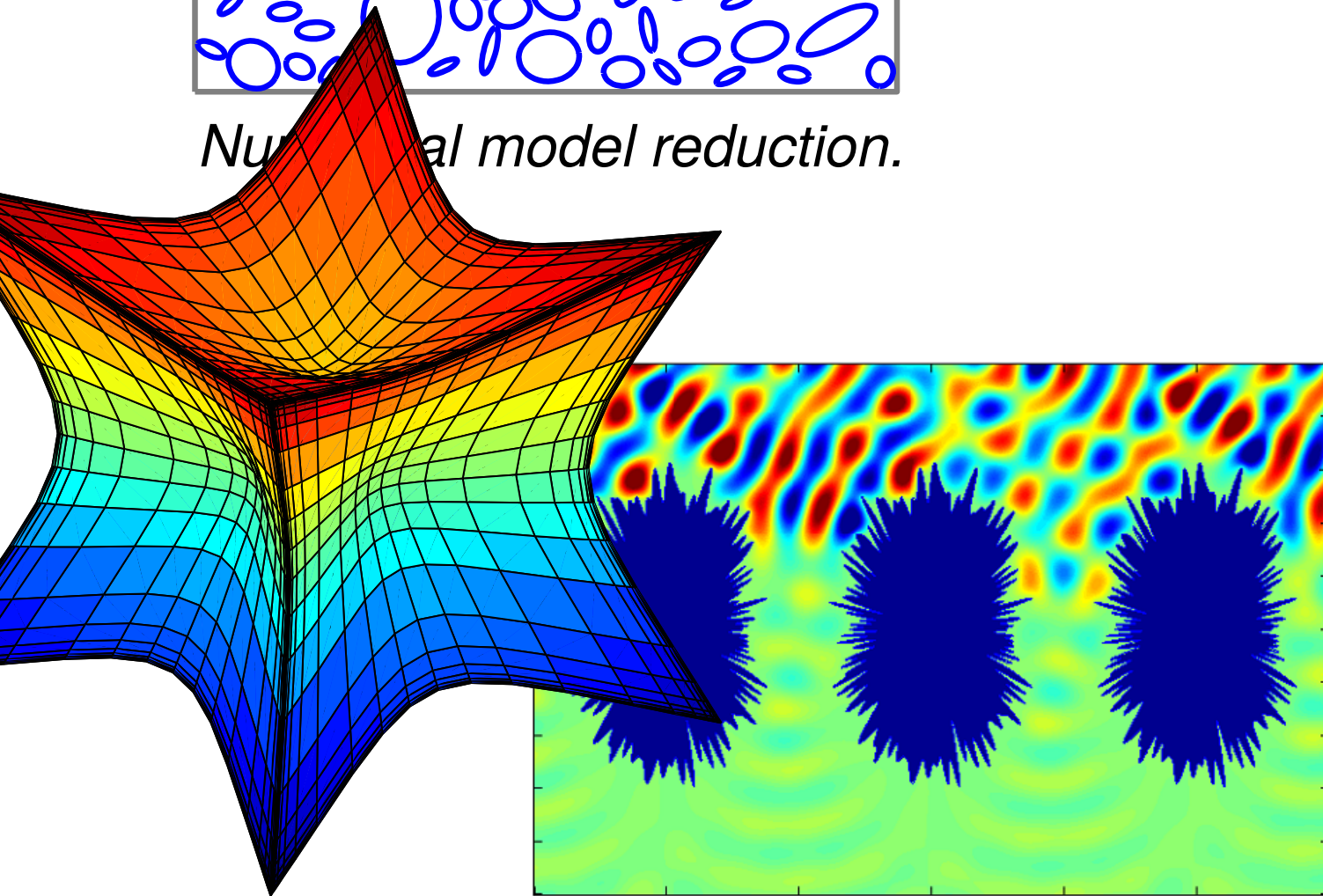


Accurate discretization of corners and edges.

Advantages of direct solvers over iterative solvers, subtleties:

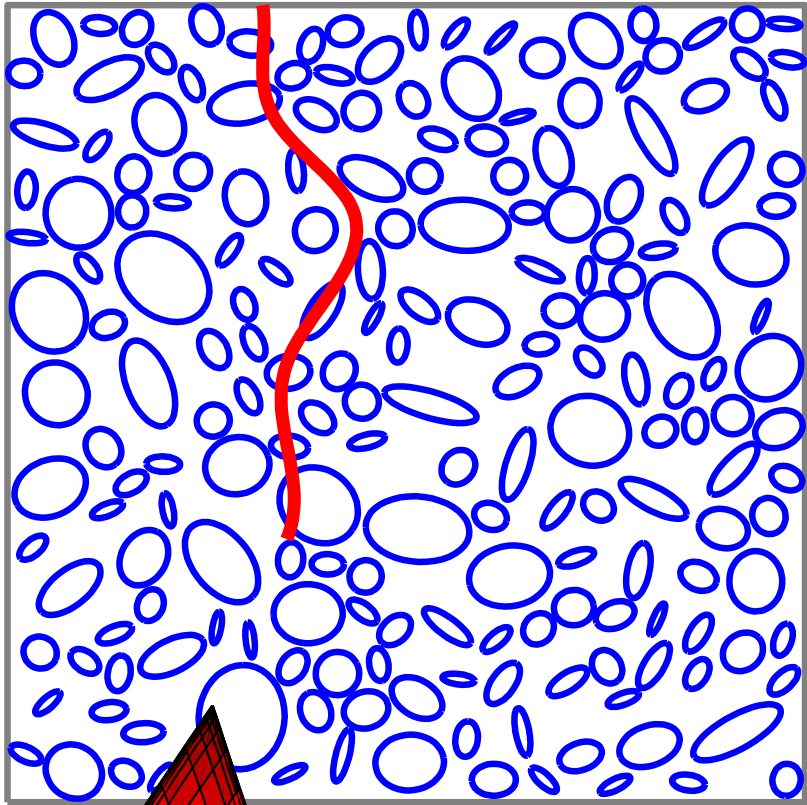


Numerical model reduction.

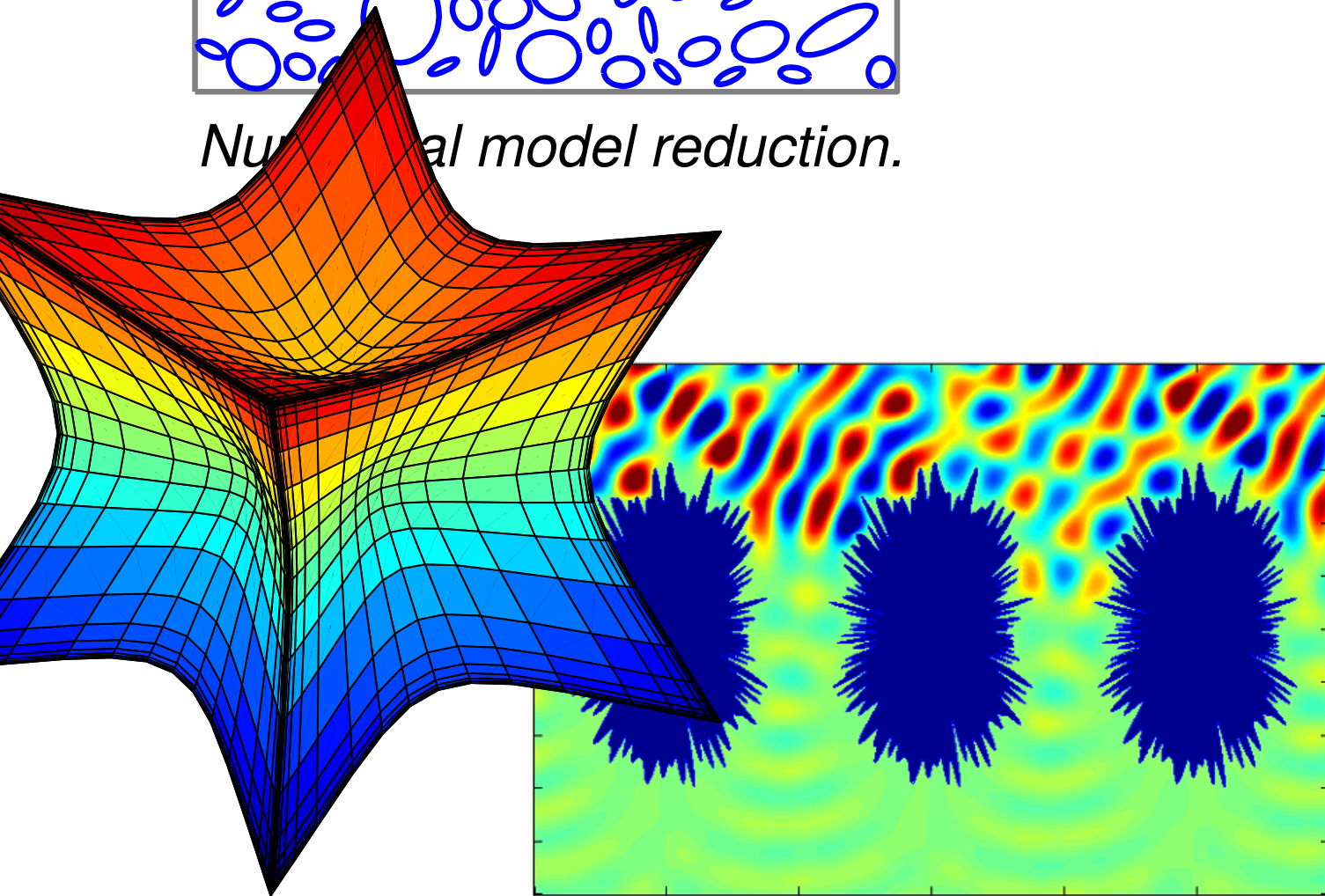


Accurate discretization of corners and edges.

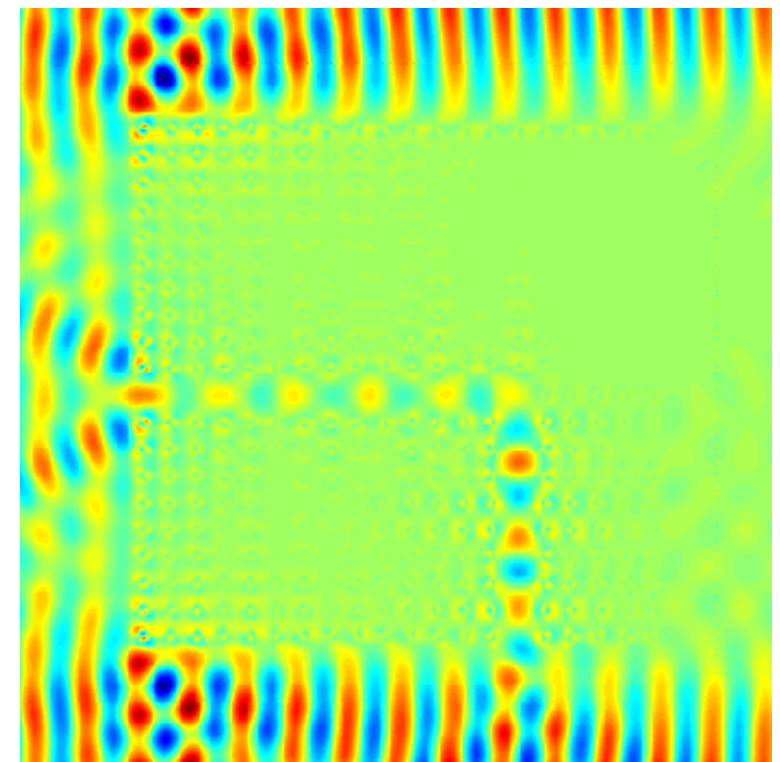
Advantages of direct solvers over iterative solvers, subtleties:



Numerical model reduction.

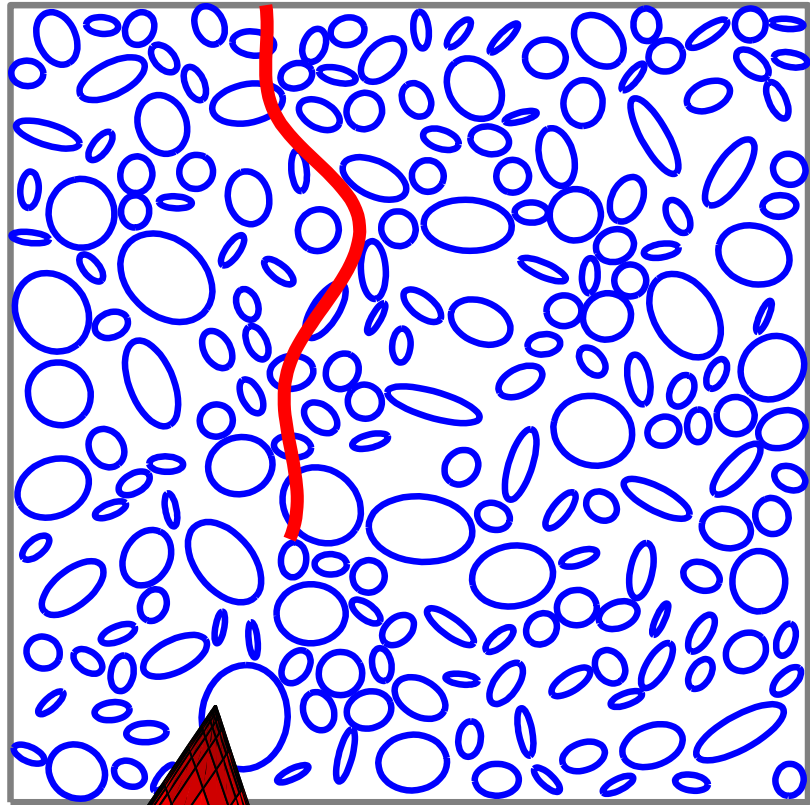


Accurate discretization of corners and edges.

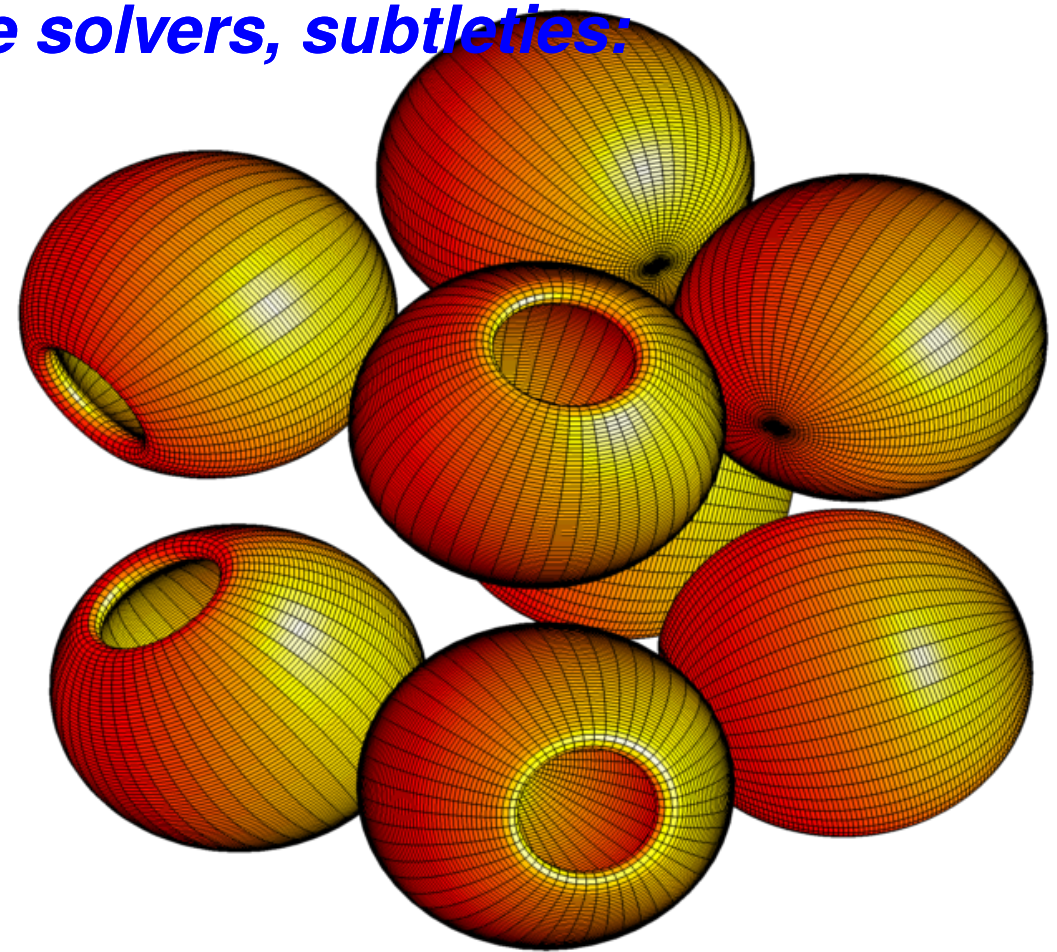


*FEM/BEM coupling. Operator algebra.
“Gluing” operators together.*

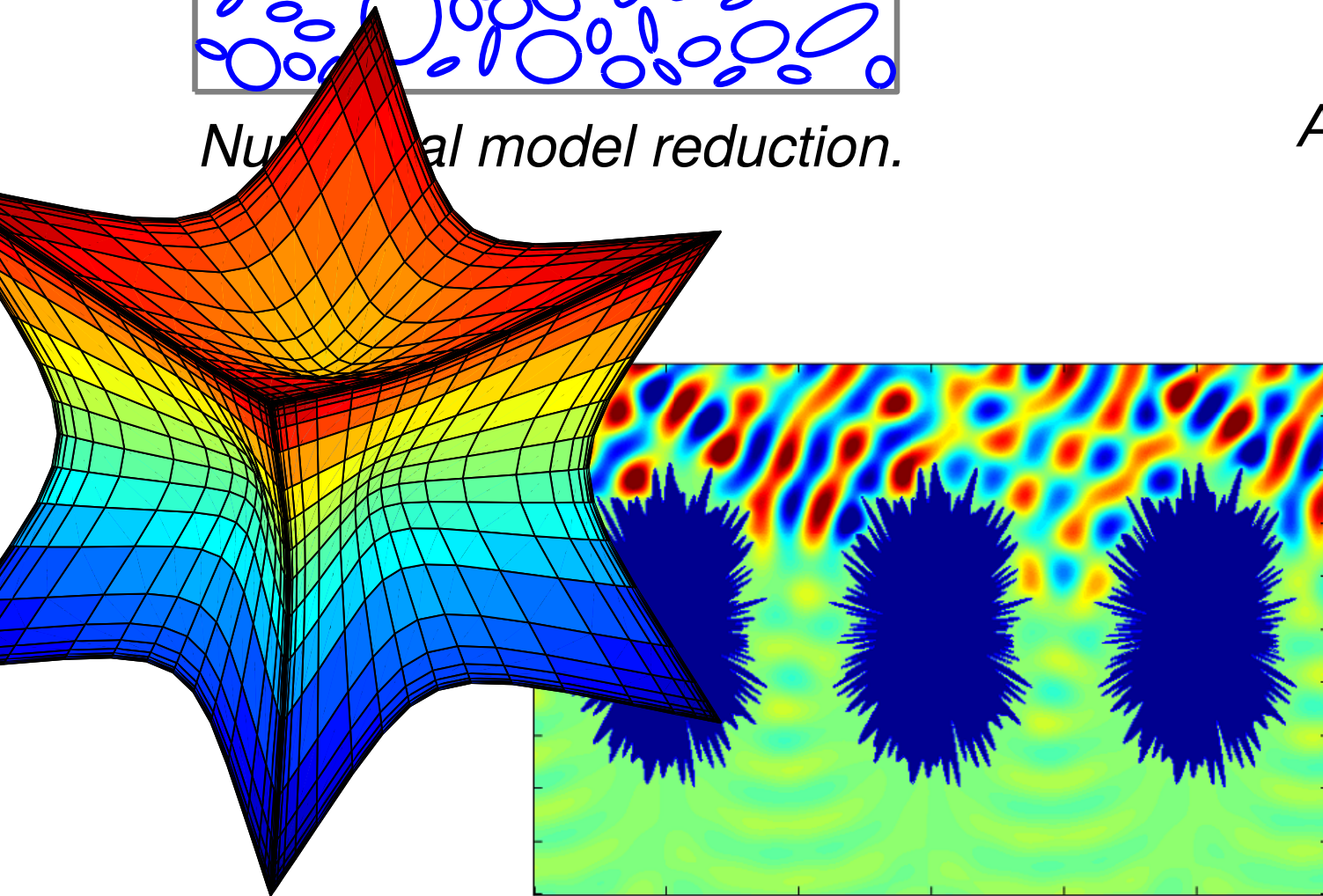
Advantages of direct solvers over iterative solvers, subtleties:



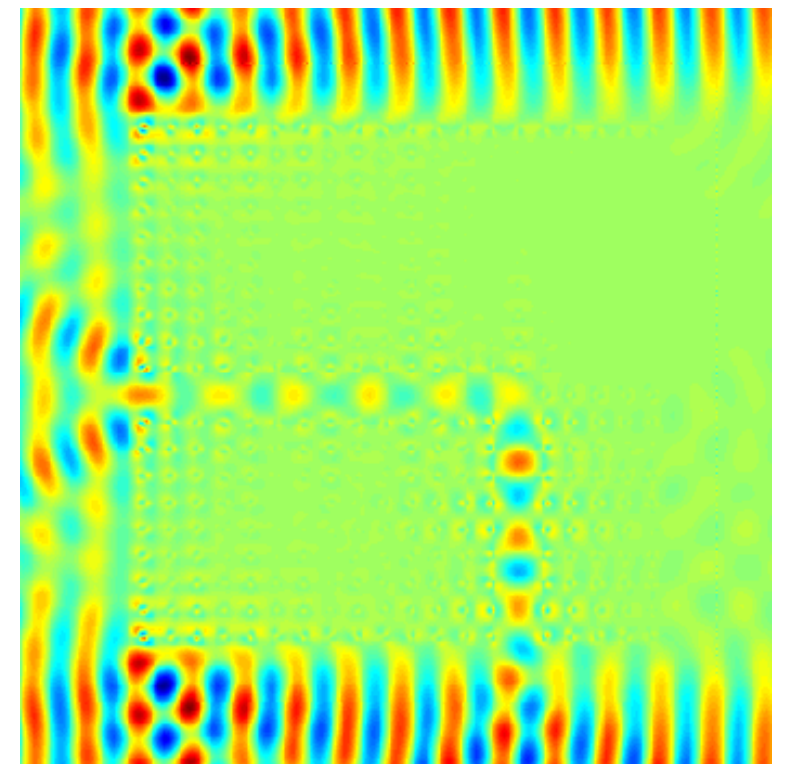
Numerical model reduction.



Accelerate iterative solvers.



Accurate discretization of corners and edges.



*FEM/BEM coupling. Operator algebra.
"Gluing" operators together.*

Fast inversion of discretized integral equations – why it works

How is it that you can invert a dense matrix of size $n \times n$ in less than $O(n^3)$ flops?

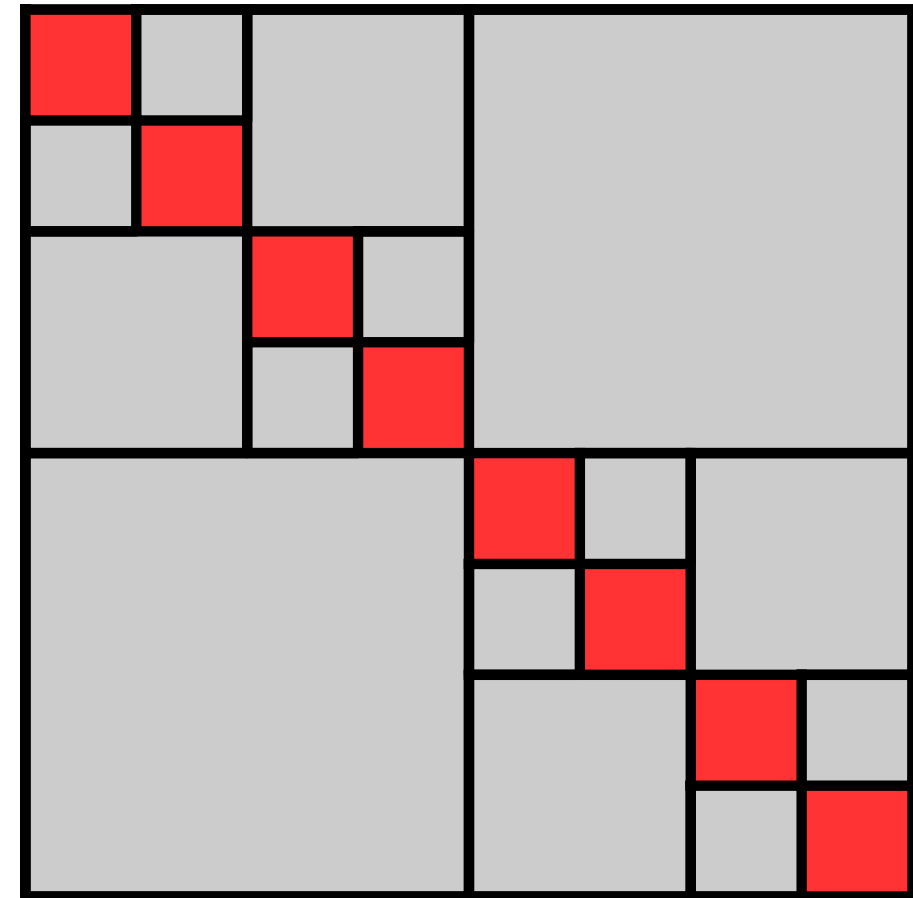
- The core idea: Exploit rank deficiencies in the coefficient matrix.
- Inversion of matrices with rank structure.
- Connection between integral equation formulations and sparse direct solvers.

Consider the problem of solving an integral equations such as, e.g.,

$$c q(\mathbf{x}) + \int_{\Psi} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) dS(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Psi. \quad (\text{IE})$$

Core idea: Upon discretization, (IE) leads to a matrix with *off-diagonal blocks of low numerical rank*. (Simplistic example shown to the right.)

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.

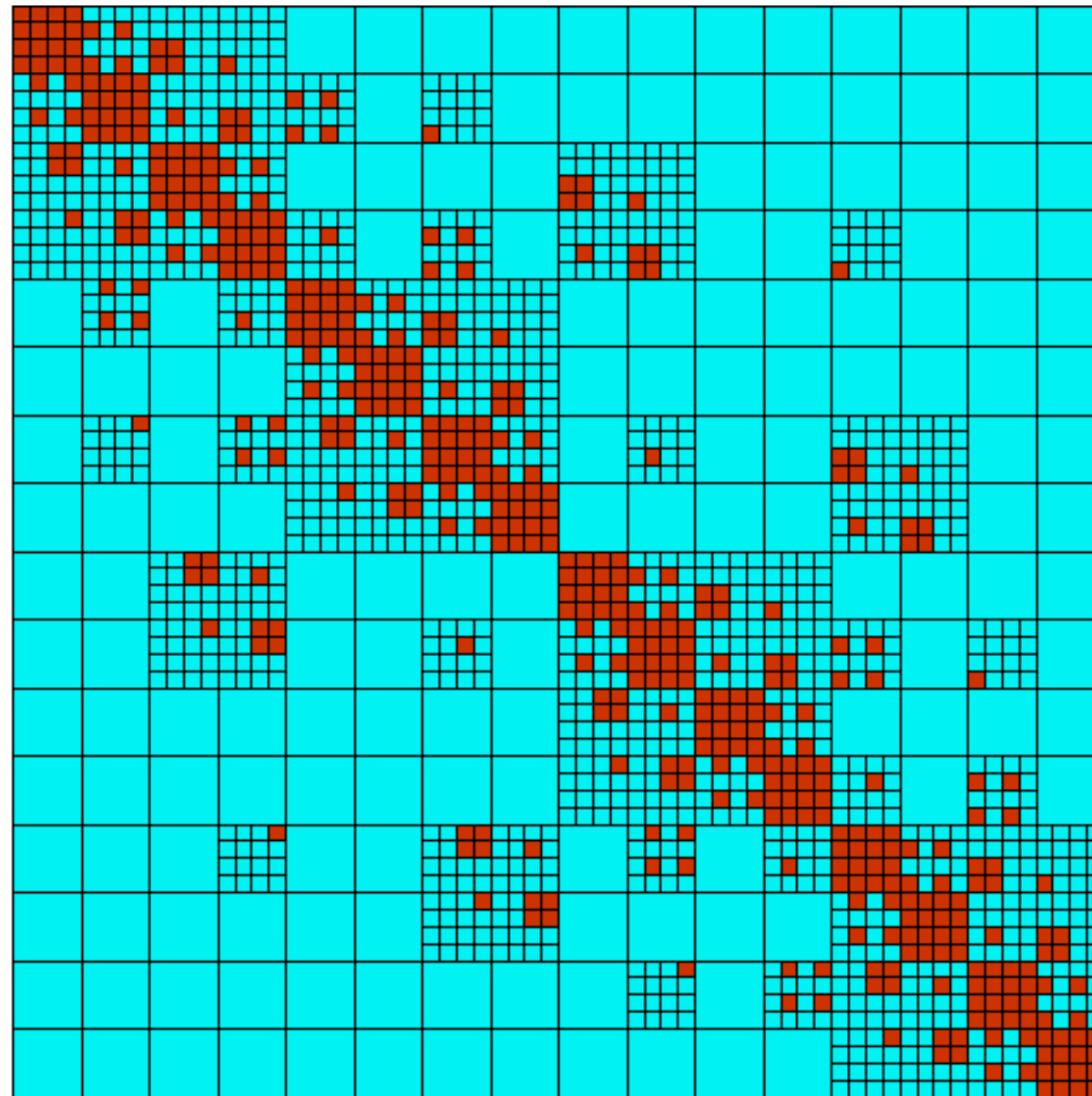
Strong connections to Calderón-Zygmund theory for singular integral operators.

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, ...); ...

Consider the problem of solving an integral equations such as, e.g.,

$$c q(\mathbf{x}) + \int_{\Psi} k(\mathbf{x}, \mathbf{y}) q(\mathbf{y}) dS(\mathbf{y}) = f(\mathbf{x}), \quad \mathbf{x} \in \Psi. \quad (\text{IE})$$

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



Inversion of structured matrices: Identity plus low rank

As a warm-up, let us consider a matrix of the form *identity + low rank*.

To be precise, suppose that a dense $n \times n$ matrix \mathbf{A} is invertible, and that for some $k \ll n$, it admits the representation

$$\mathbf{A} = \mathbf{I} + \mathbf{U} \mathbf{V}^*.$$

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

Inversion of structured matrices: Identity plus low rank

As a warm-up, let us consider a matrix of the form *identity + low rank*.

To be precise, suppose that a dense $n \times n$ matrix \mathbf{A} is invertible, and that for some $k \ll n$, it admits the representation

$$\begin{array}{ccccccc} \mathbf{A} & = & \mathbf{I} & + & \mathbf{U} & \mathbf{V}^* & . \\ n \times n & & n \times n & & n \times k & k \times n & \end{array}$$

While \mathbf{A} is dense, it has *rank structure* that allows us to:

- Store it using $O(nk)$ floats.
- Execute the matrix vector multiplication in $O(nk)$ flops.

$O(n^2)$ in general case.

$O(n^2)$ in general case.

Inversion of structured matrices: Identity plus low rank

As a warm-up, let us consider a matrix of the form *identity + low rank*.

To be precise, suppose that a dense $n \times n$ matrix \mathbf{A} is invertible, and that for some $k \ll n$, it admits the representation

$$\mathbf{A} = \mathbf{I} + \mathbf{U} \mathbf{V}^*.$$

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

Then the inverse of \mathbf{A} takes the form

(W)
$$\mathbf{A}^{-1} = \mathbf{I} - \mathbf{U} (\mathbf{I} + \mathbf{V}^* \mathbf{U})^{-1} \mathbf{V}^*.$$

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

Using (W), we can form \mathbf{A}^{-1} (implicitly) in $O(nk^2)$ work, instead of $O(n^3)$.

Inversion of structured matrices: Identity plus low rank

As a warm-up, let us consider a matrix of the form *identity + low rank*.

To be precise, suppose that a dense $n \times n$ matrix \mathbf{A} is invertible, and that for some $k \ll n$, it admits the representation

$$\mathbf{A} = \mathbf{I} + \mathbf{U} \mathbf{V}^*.$$

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

Then the inverse of \mathbf{A} takes the form

(W)
$$\mathbf{A}^{-1} = \mathbf{I} - \mathbf{U} (\mathbf{I} + \mathbf{V}^* \mathbf{U})^{-1} \mathbf{V}^*.$$

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

Using (W), we can form \mathbf{A}^{-1} (implicitly) in $O(nk^2)$ work, instead of $O(n^3)$.

There are many versions of Woodbury!

For instance, consider a matrix of the form *“easy to invert” + low rank*.

To be precise, suppose that \mathbf{A} is invertible, that

$$\mathbf{A} = \mathbf{D} + \mathbf{U} \mathbf{V}^*,$$

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

and that you can easily compute \mathbf{D}^{-1} . Then \mathbf{A}^{-1} can be constructed via

$$\mathbf{A}^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1} \mathbf{U} (\mathbf{I} + \mathbf{V}^* \mathbf{D}^{-1} \mathbf{U})^{-1} \mathbf{V}^* \mathbf{D}^{-1}.$$

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

Inversion of structured matrices: Tridiagonal

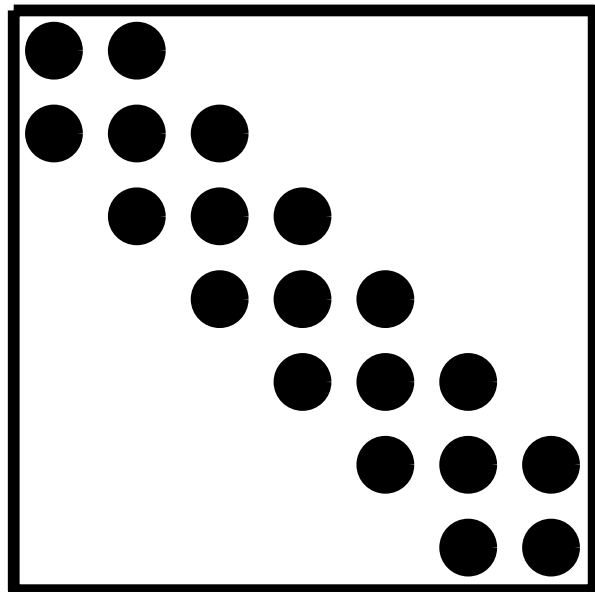
Consider a simple 2-point BVP on the interval $[0, 1]$:

$$(BVP) \quad \begin{cases} -\frac{d^2u}{dx^2} + p(x) \frac{du(x)}{dx} + m(x)u(x) = g(x), & x \in (0, 1), \\ u(0) = f_L, \\ u(1) = f_R. \end{cases}$$

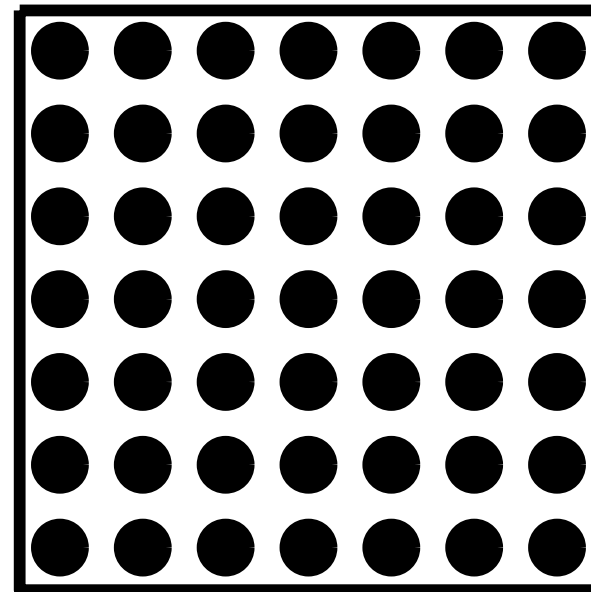
Discretizing (BVP) using a standard second order finite difference scheme, we get

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

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



Sparsity pattern of \mathbf{A} .



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

Inversion of structured matrices: Tridiagonal

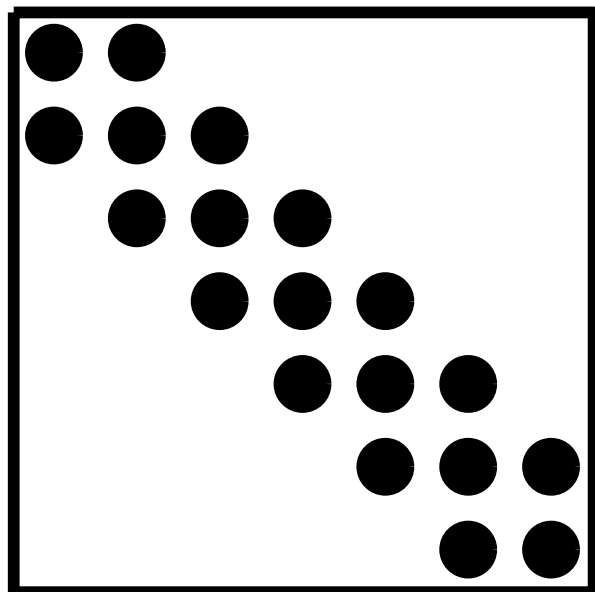
Consider a simple 2-point BVP on the interval $[0, 1]$:

$$(BVP) \quad \begin{cases} -\frac{d^2u}{dx^2} + p(x) \frac{du(x)}{dx} + m(x)u(x) = g(x), & x \in (0, 1), \\ u(0) = f_L, \\ u(1) = f_R. \end{cases}$$

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

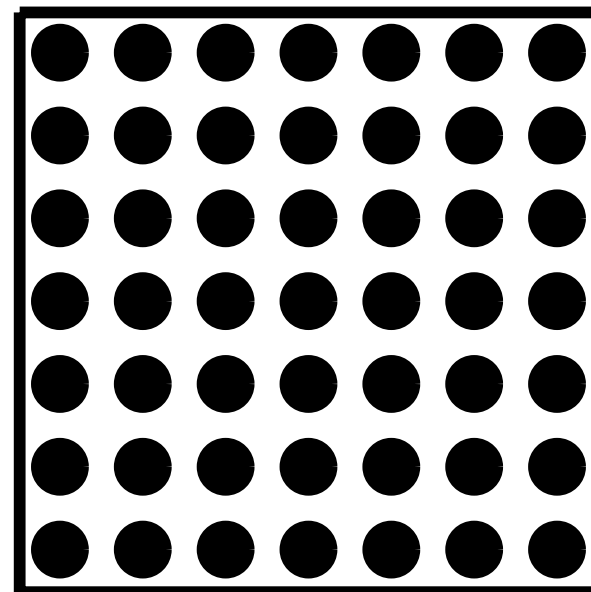
$$\mathbf{A}\mathbf{u} = \mathbf{b},$$

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.



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

Inversion of structured matrices: Tridiagonal

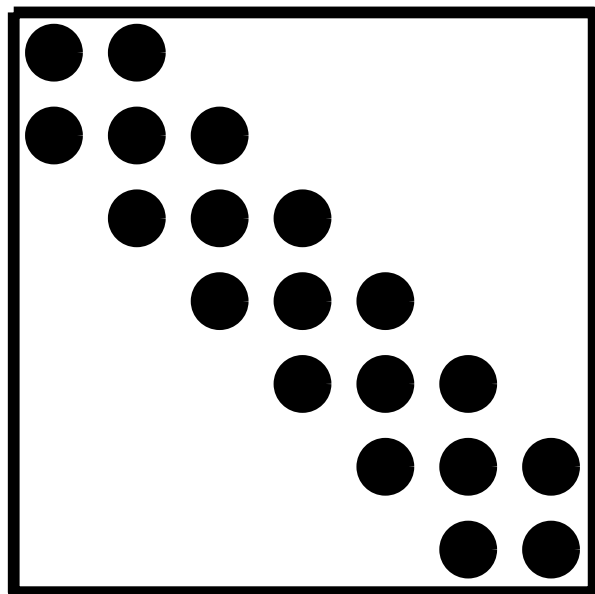
Consider a simple 2-point BVP on the interval $[0, 1]$:

$$(BVP) \quad \begin{cases} -\frac{d^2u}{dx^2} + p(x) \frac{du(x)}{dx} + m(x)u(x) = g(x), & x \in (0, 1), \\ u(0) = f_L, \\ u(1) = f_R. \end{cases}$$

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

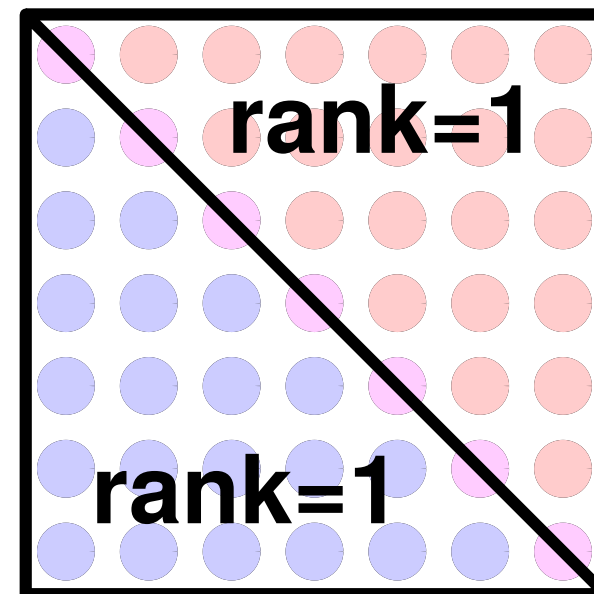
$$\mathbf{A}u = \mathbf{b},$$

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.



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

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

Inversion of structured matrices: Tridiagonal

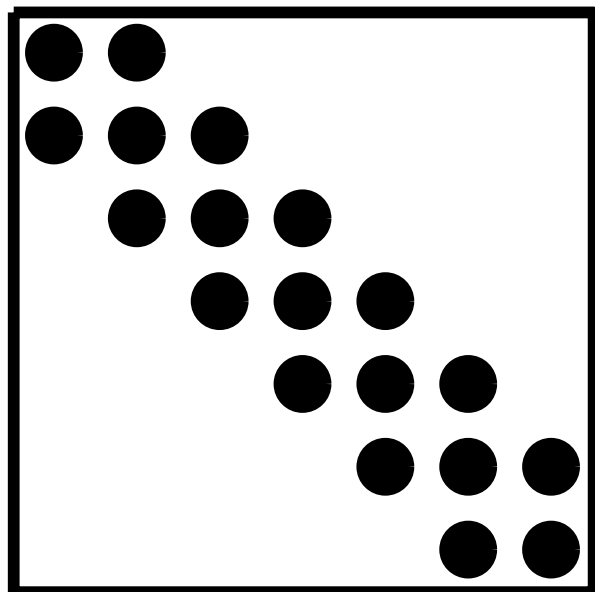
Consider a simple 2-point BVP on the interval $[0, 1]$:

$$(BVP) \quad \begin{cases} -\frac{d^2u}{dx^2} + p(x) \frac{du(x)}{dx} + m(x)u(x) = g(x), & x \in (0, 1), \\ u(0) = f_L, \\ u(1) = f_R. \end{cases}$$

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

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

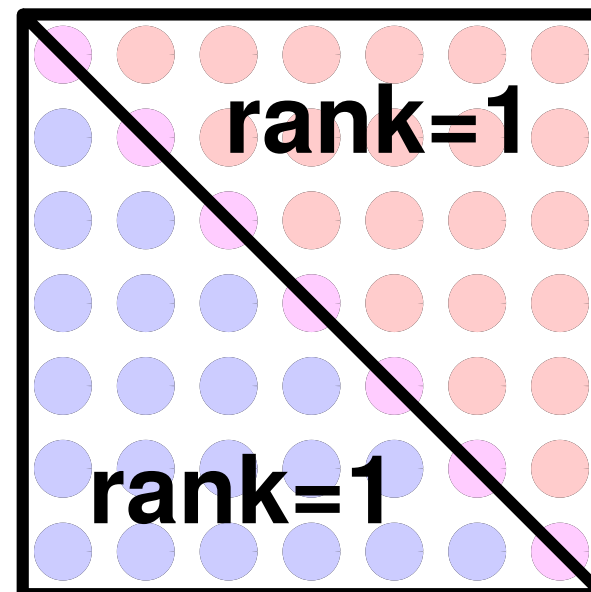
where \mathbf{A} is a sparse matrix of size, say, $n \times n$. Then \mathbf{A}^{-1} is dense.



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 $[0, 1]$:

$$(BVP) \quad \left\{ \begin{array}{l} -\frac{d^2u}{dx^2} + p(x) \frac{du(x)}{dx} + m(x)u(x) = g(x), \quad x \in (0, 1), \\ u(0) = f_L, \\ u(1) = f_R. \end{array} \right.$$

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.

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.)

Inversion of structured matrices: Semi-separable plus diagonal

Let us again consider a two point boundary value problem

$$(BVP) \quad -u''(y) + m(y)u(y) = g(y), \quad y \in (0, 1),$$

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

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

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

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

where

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

Fact 1: The equation (BVP) is equivalent to

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

Inversion of structured matrices: Semi-separable plus diagonal

Let us again consider a two point boundary value problem

$$(BVP) \quad -u''(y) + m(y)u(y) = g(y), \quad y \in (0, 1),$$

now with zero boundary data.

Fact 1: The equation (BVP) is equivalent to

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

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

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

Inversion of structured matrices: Semi-separable plus diagonal

Let us again consider a two point boundary value problem

$$(BVP) \quad -u''(y) + m(y)u(y) = g(y), \quad y \in (0, 1),$$

now with zero boundary data.

Fact 1: The equation (BVP) is equivalent to

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

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

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

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

$$(\mathbf{I} + \mathbf{GM})\mathbf{u} = \mathbf{Gg}.$$

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

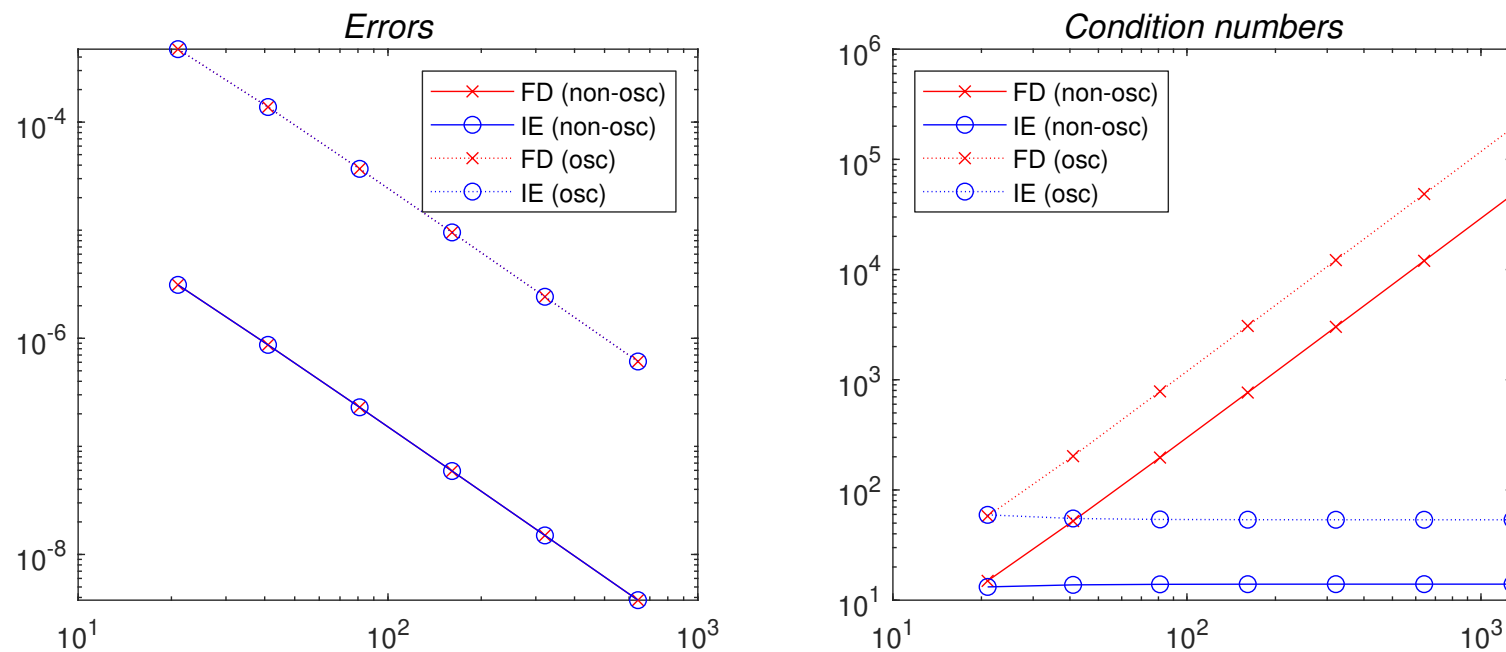
How do the two methods compare?

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?



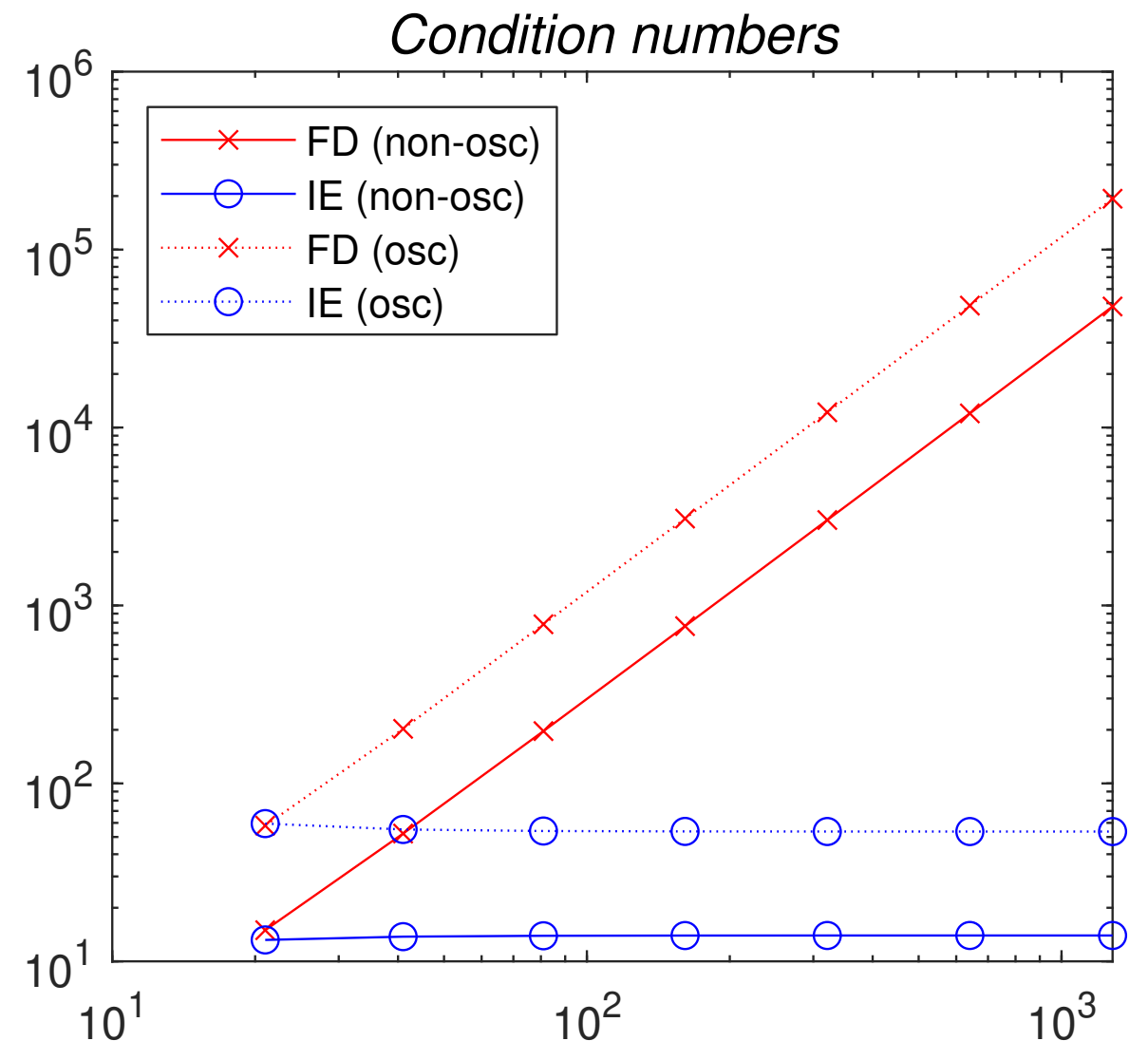
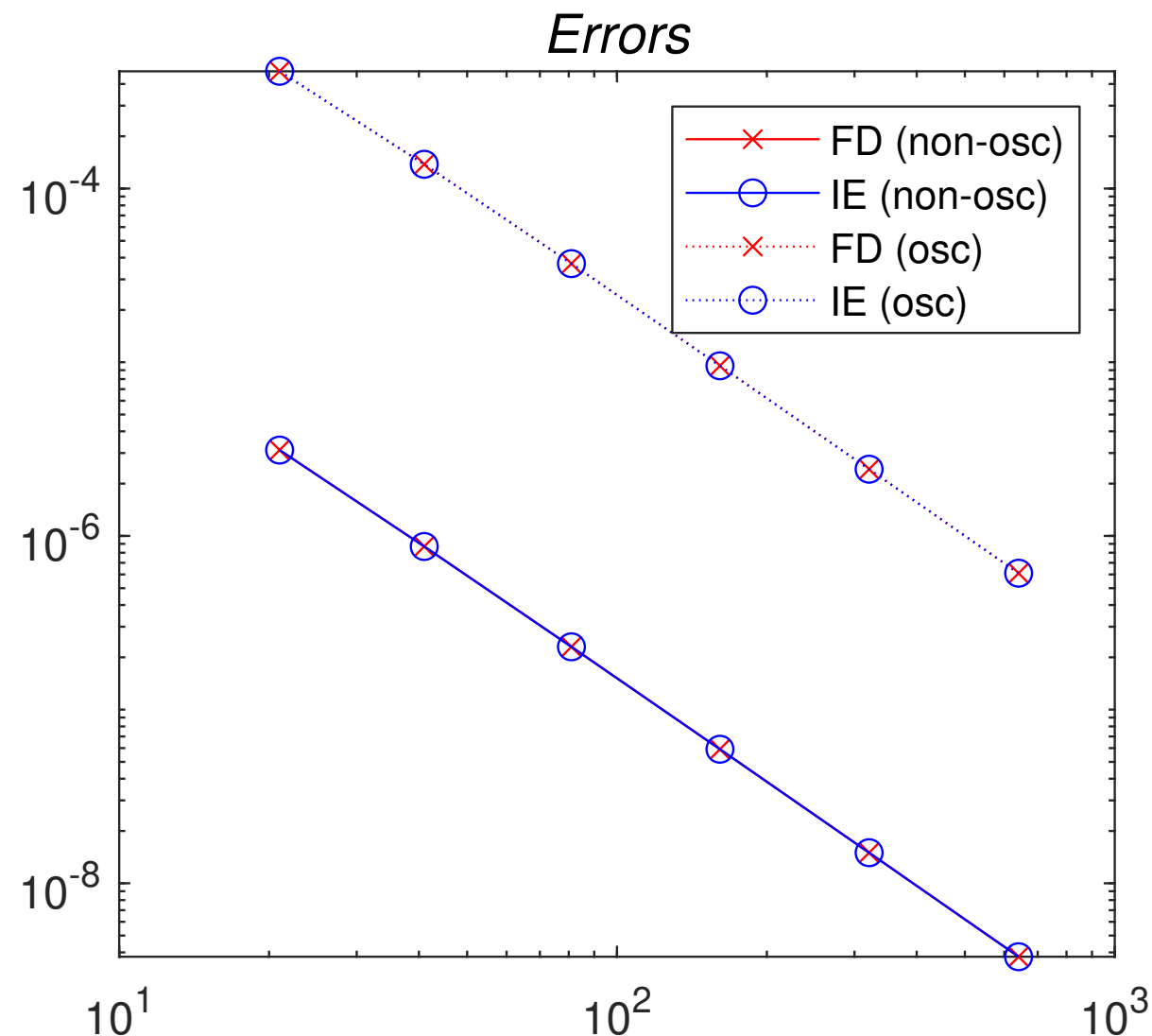
We considered a non-oscillatory problem “(non-osc)” where $m(x) = 100(1 + x) \cos(x)$ and $g(x) = 1 + \cos(1 + x)$. We then swapped the sign of m (but kept everything else the same) to get a problem with an oscillatory solution “(osc)”. The left plot shows the errors incurred (in max norm), while the 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!

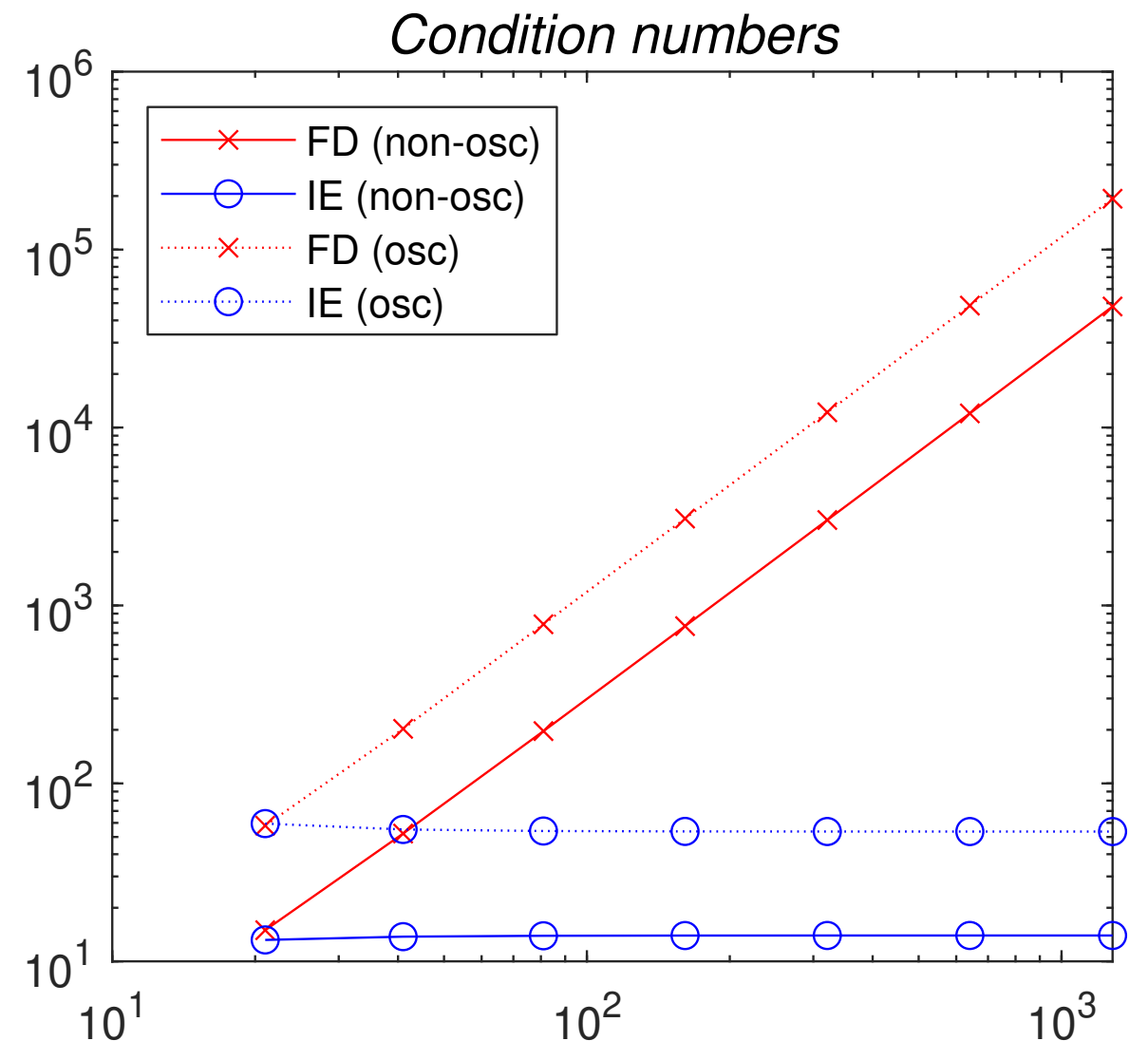
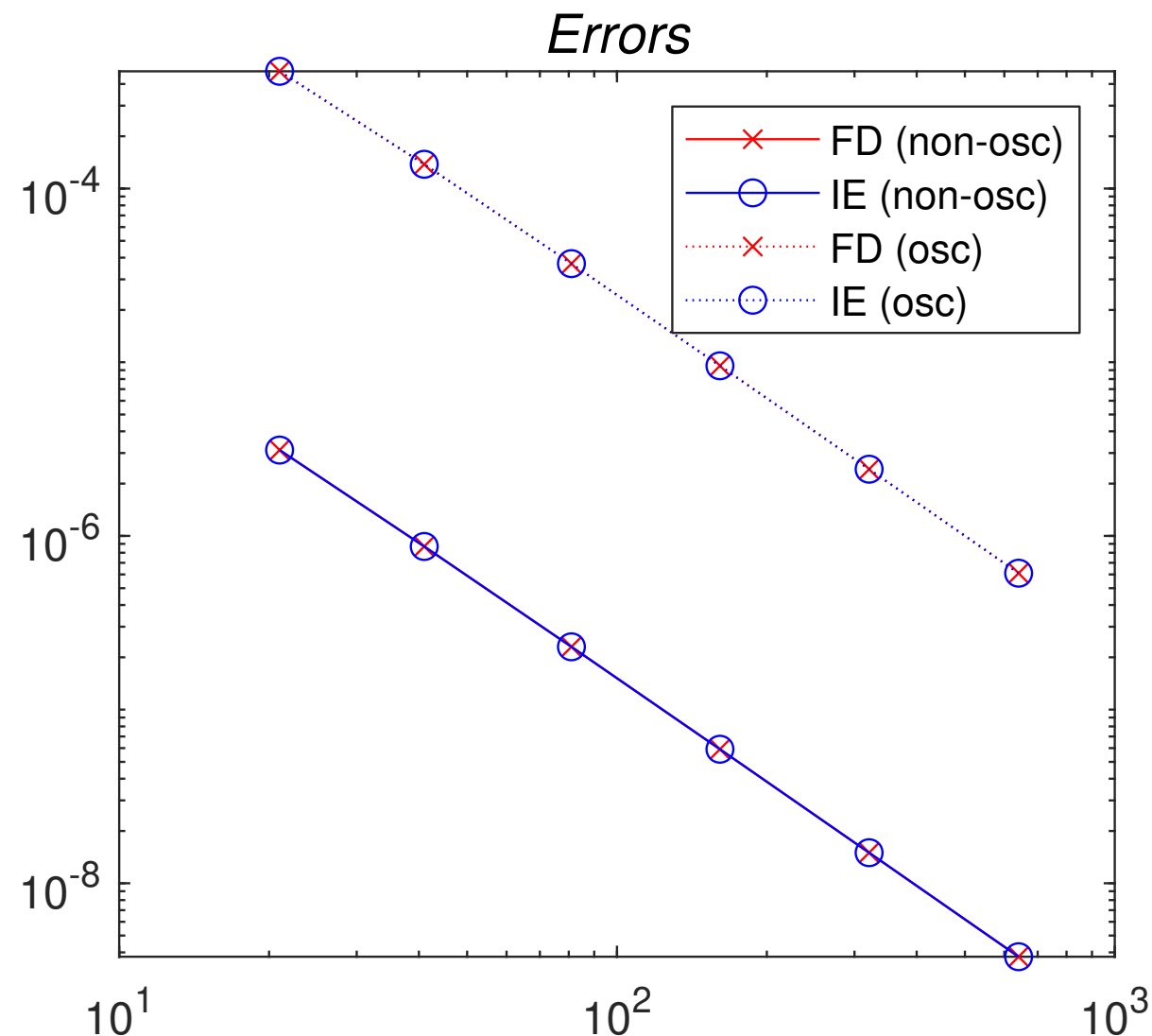
(Fun fact: The two methods are mathematically equivalent – errors are *identical*!)

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?

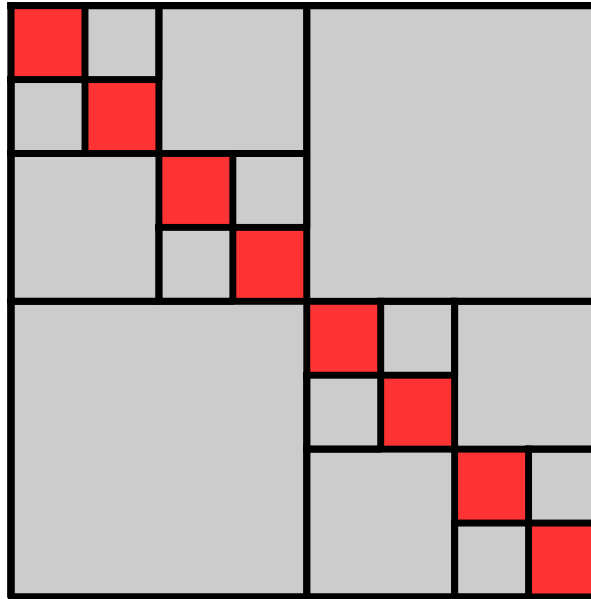
What about computational costs?

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

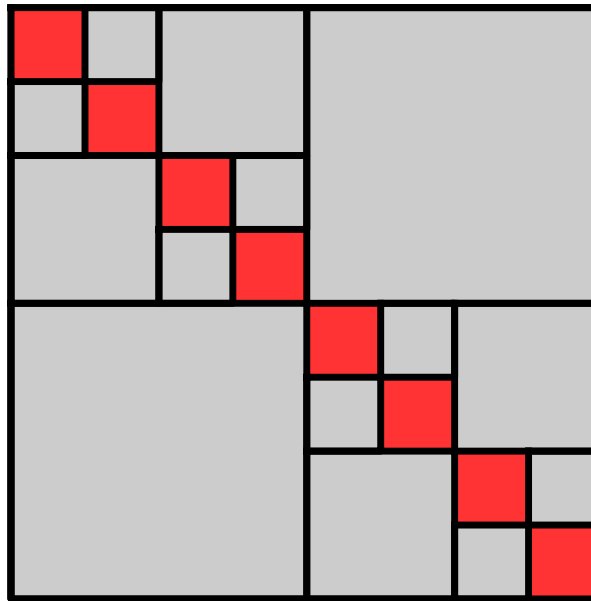
Now let us consider a slightly more complex structure:



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.

We will today show a very simple *recursive* inversion procedure.

(A much better algorithm will be described on Wednesday.)

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

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


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


Inversion of structured matrices: HODLR

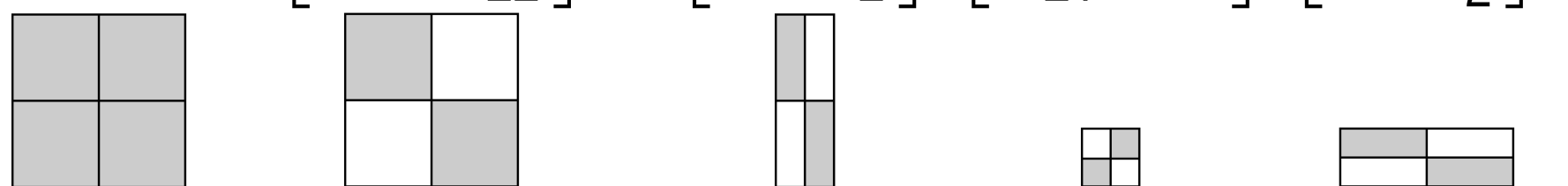
We seek to invert a matrix \mathbf{A} as shown. Each block is of size $n \times n$, and \mathbf{A}_{12} and \mathbf{A}_{21} have rank $k < n$.

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

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

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


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

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


Inversion of structured matrices: HODLR

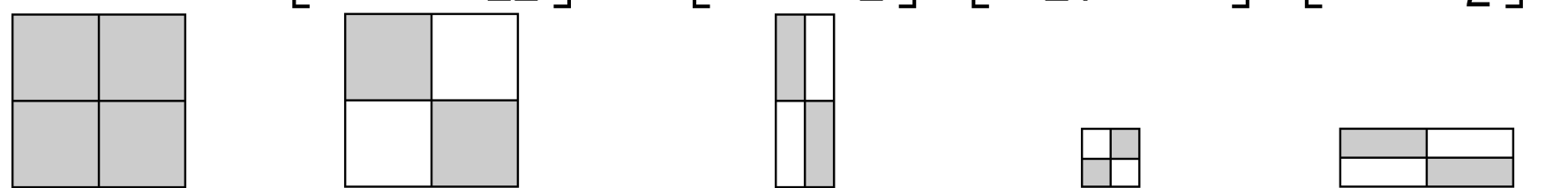
We seek to invert a matrix \mathbf{A} as shown. Each block is of size $n \times n$, and \mathbf{A}_{12} and \mathbf{A}_{21} have rank $k < n$.

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

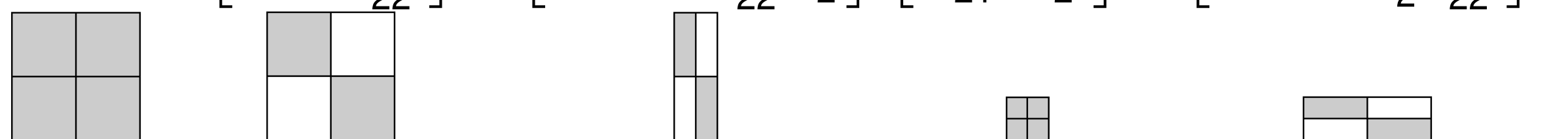
We first form low-rank factorizations of \mathbf{A}_{12} and \mathbf{A}_{21} so that

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


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

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

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

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

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

$2n \times 2n \qquad 2n \times 2n \qquad 2n \times 2k \qquad 2k \times 2k \qquad 2k \times 2n$

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

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

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

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

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

$2n \times 2n \qquad 2n \times 2n \qquad 2n \times 2k \qquad 2k \times 2k \qquad 2k \times 2n$

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

Connection between BIE and sparse direct solvers

Surprise!

Connection between BIE and sparse direct solvers

Surprise!

Fast direct solver methodology developed for BIEs has proven useful for accelerating *sparse direct solvers* to linear complexity!

Connection between BIE and sparse direct solvers

Surprise!

Fast direct solver methodology developed for BIEs has proven useful for accelerating *sparse direct solvers* to linear complexity!

To illustrate, let us consider what happens if we directly discretize the PDE (using, say, finite elements or finite differences) to obtain a linear system

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

involving a *sparse* coefficient matrix \mathbf{A} .

Key idea: Do a sparse LU factorization based on a “nested dissection” ordering of the grid as an outer solver. Then use rank structured matrix algebra to deal with the dense matrices that arise.

Connection between BIE and sparse direct solvers

Let $\Omega = [0, 1]^2$ and $\Gamma = \partial\Omega$. We seek to solve

$$(13) \quad \begin{cases} -\Delta u(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma. \end{cases}$$

We introduce an $n \times n$ grid on Ω with nodes $\{\mathbf{x}_j\}_{j=1}^N$ where $N = n^2$, see Figure A. Letting $\mathbf{u} = [\mathbf{u}(j)]_{j=1}^N$ denote a vector of approximate solution values, $\mathbf{u}(j) \approx u(\mathbf{x}_j)$, and using the standard five-point stencil to discretize $-\Delta$, we end up with a sparse linear system

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

where $[\mathbf{A}\mathbf{u}](k) = \frac{1}{h^2}(4\mathbf{u}(k) - \mathbf{u}(k_s) - \mathbf{u}(k_e) - \mathbf{u}(k_n) - \mathbf{u}(k_w))$, see Figure B.

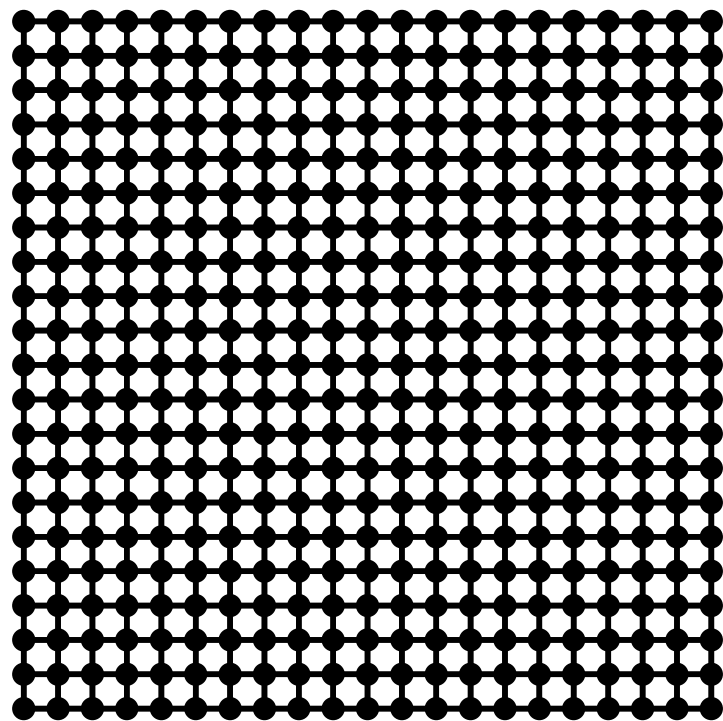


Figure A: The grid

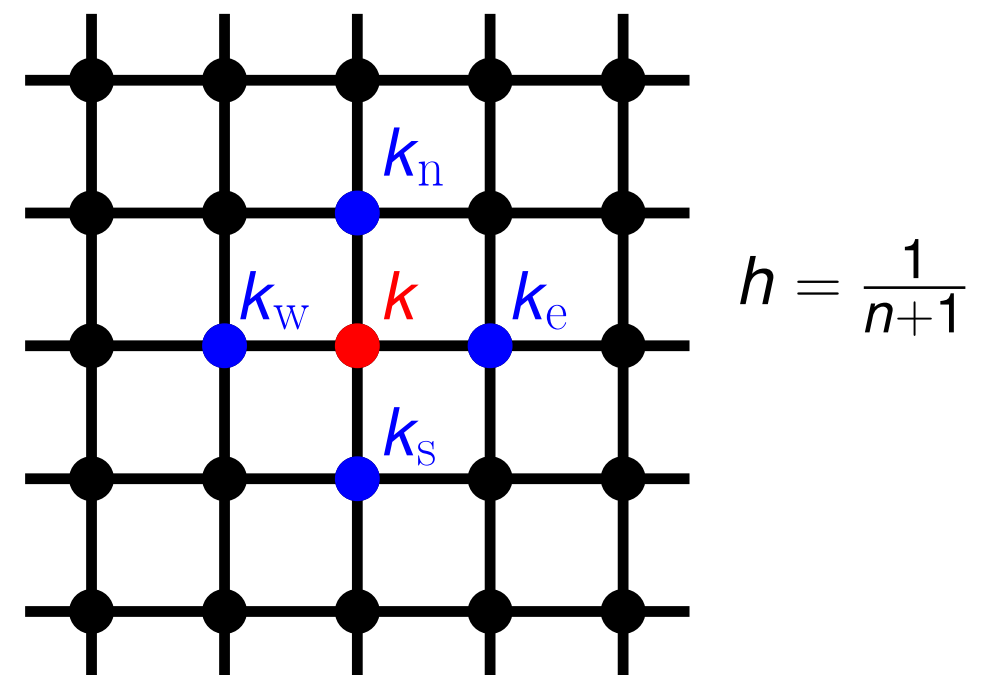
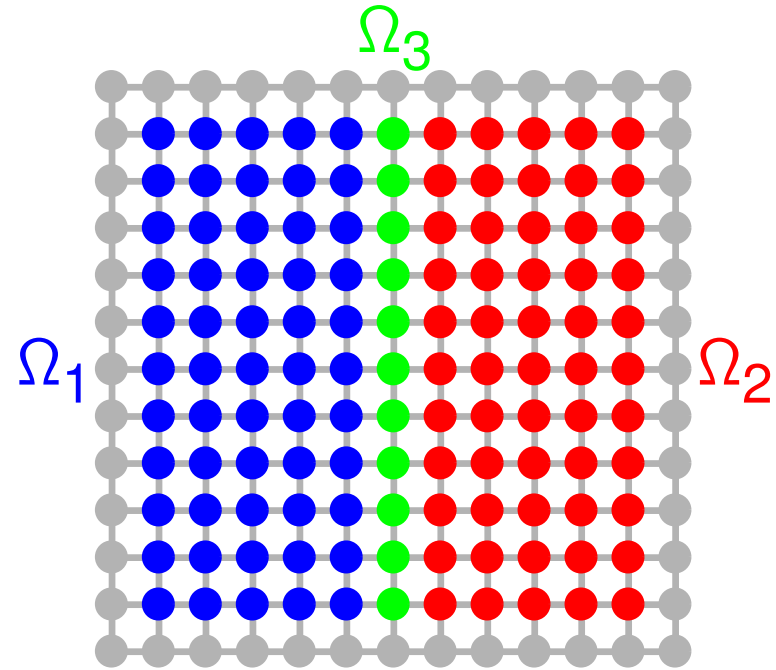


Figure B: The 5-point stencil

Connection between BIE and sparse direct solvers

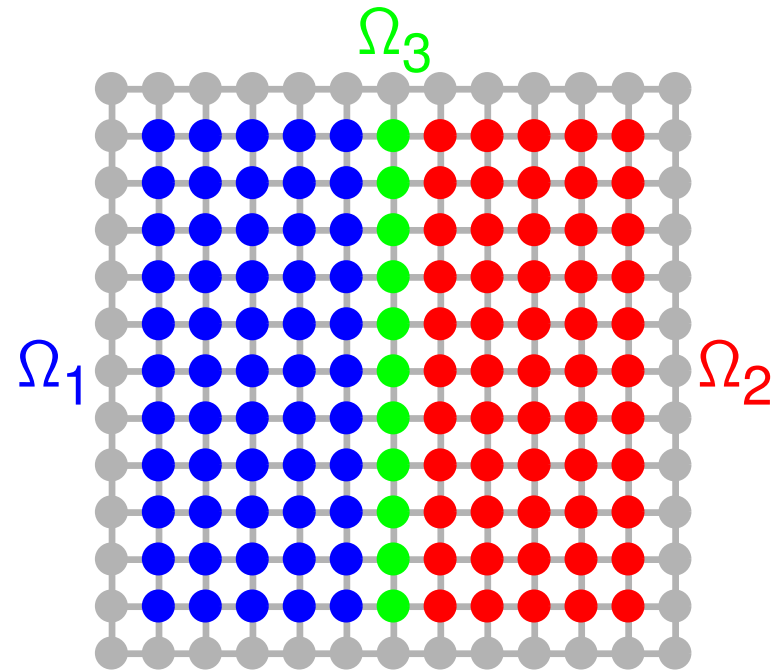
Divide-and-conquer: Split the nodes in three groups as shown so that there are no connections between nodes in Ω_1 and Ω_2 . Then \mathbf{A} has zero blocks as shown:



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

Connection between BIE and sparse direct solvers

Divide-and-conquer: Split the nodes in three groups as shown so that there are no connections between nodes in Ω_1 and Ω_2 . Then \mathbf{A} has zero blocks as shown:



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

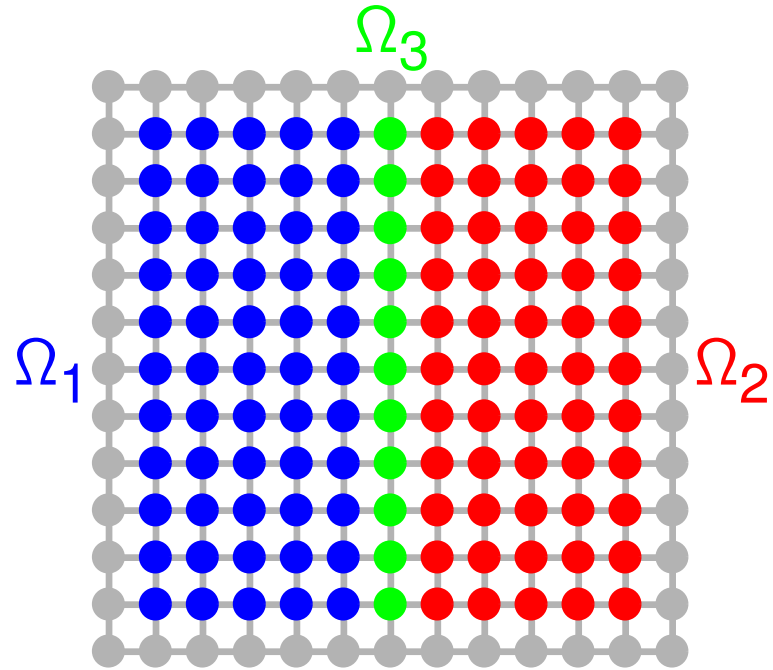
Now suppose that we can somehow construct \mathbf{A}_{11}^{-1} and \mathbf{A}_{22}^{-1} . Then

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

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

Connection between BIE and sparse direct solvers

Divide-and-conquer: Split the nodes in three groups as shown so that there are no connections between nodes in Ω_1 and Ω_2 . Then \mathbf{A} has zero blocks as shown:



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

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

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

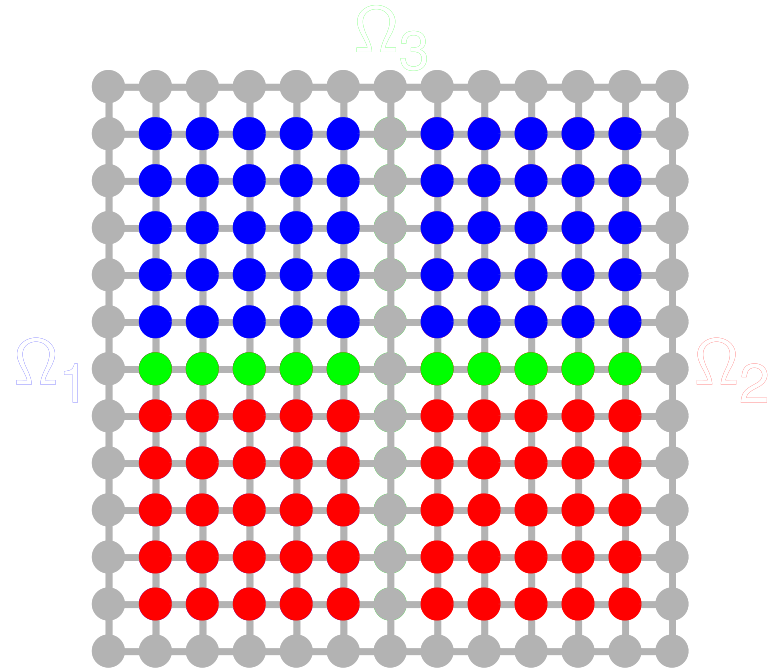
where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement*.

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

- Invert \mathbf{A}_{11} to form \mathbf{A}_{11}^{-1} . *size* $\sim N/2 \times N/2$
- Invert \mathbf{A}_{22} to form \mathbf{A}_{22}^{-1} . *size* $\sim N/2 \times N/2$
Notice the obvious recursion!
- 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* $\sim \sqrt{N} \times \sqrt{N}$

Connection between BIE and sparse direct solvers

Divide-and-conquer: Split the nodes in three groups as shown so that there are no connections between nodes in Ω_1 and Ω_2 . Then \mathbf{A} has zero blocks as shown:



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

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

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

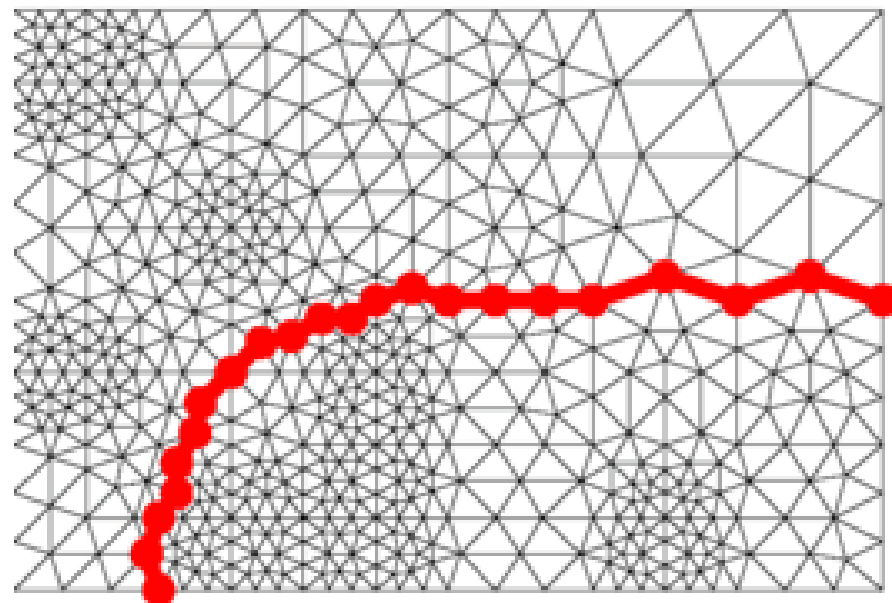
where $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$ is a *Schur complement*.

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

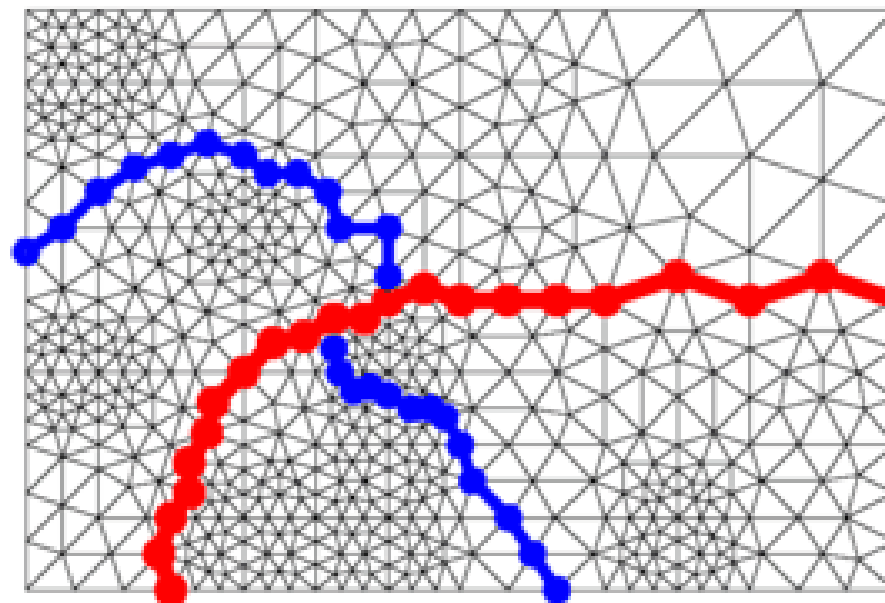
- Invert \mathbf{A}_{11} to form \mathbf{A}_{11}^{-1} . *size* $\sim N/2 \times N/2$
- Invert \mathbf{A}_{22} to form \mathbf{A}_{22}^{-1} . *size* $\sim N/2 \times N/2$
Notice the obvious recursion!
- Invert $\mathbf{S}_{33} = \mathbf{A}_{33} - \mathbf{A}_{31}\mathbf{A}_{11}^{-1}\mathbf{A}_{13} - \mathbf{A}_{32}\mathbf{A}_{22}^{-1}\mathbf{A}_{23}$.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

Connection between BIE and sparse direct solvers

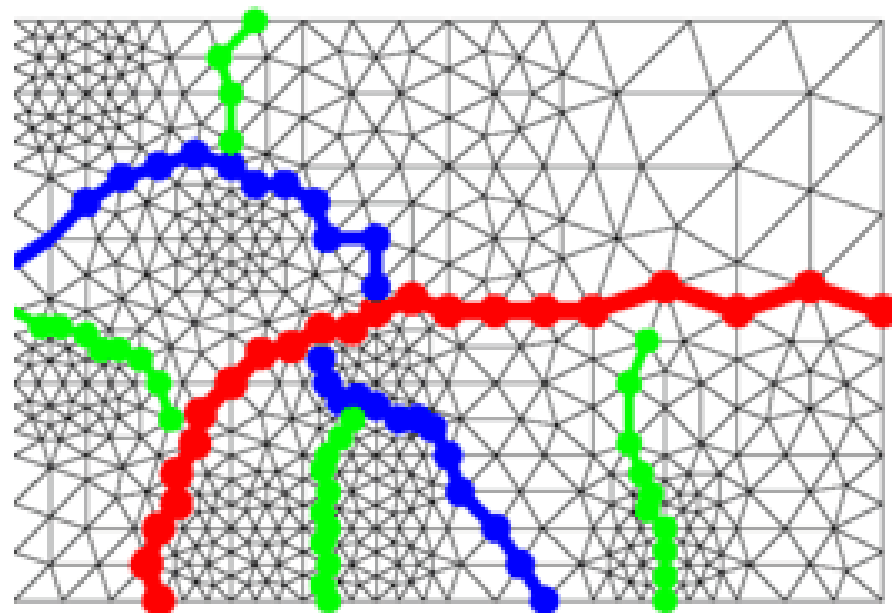
Typically, nested dissection orderings are more complicated:



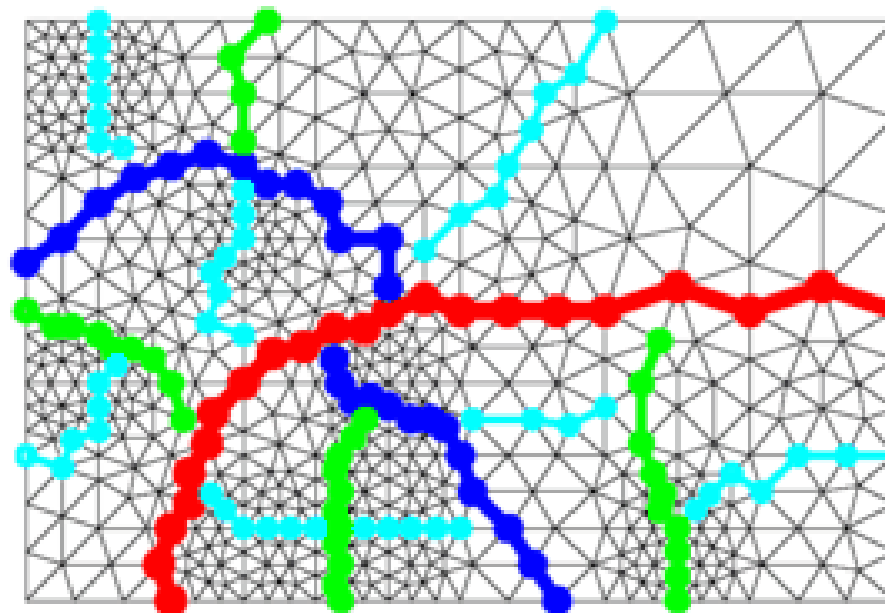
(i) *Top level separator*



(ii) *Two levels of separators*



(iii) *Three levels of separators*



(iv) *Four levels of separators*

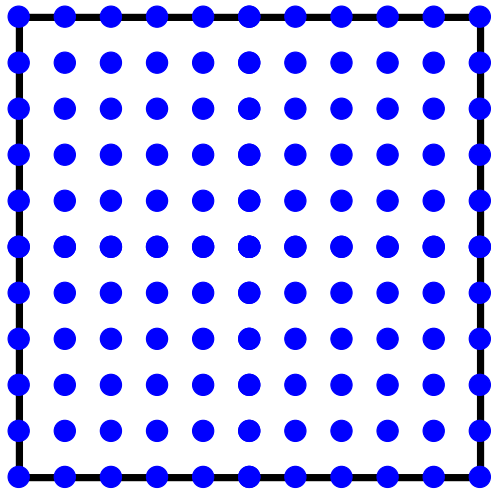
Image credit: Jianlin Xia, "Robust and Efficient Multifrontal Solver for Large Discretized PDEs", 2012

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

Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition.

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.



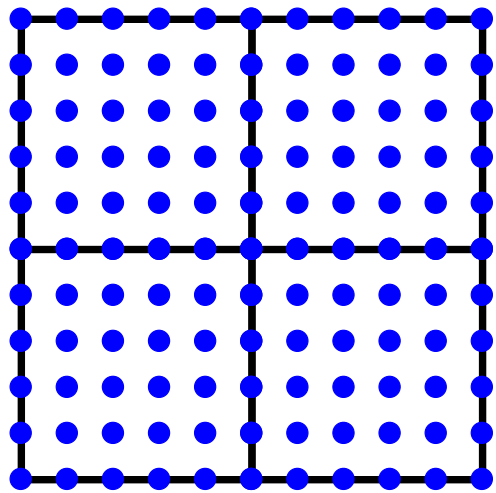
The original grid.

Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition.

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

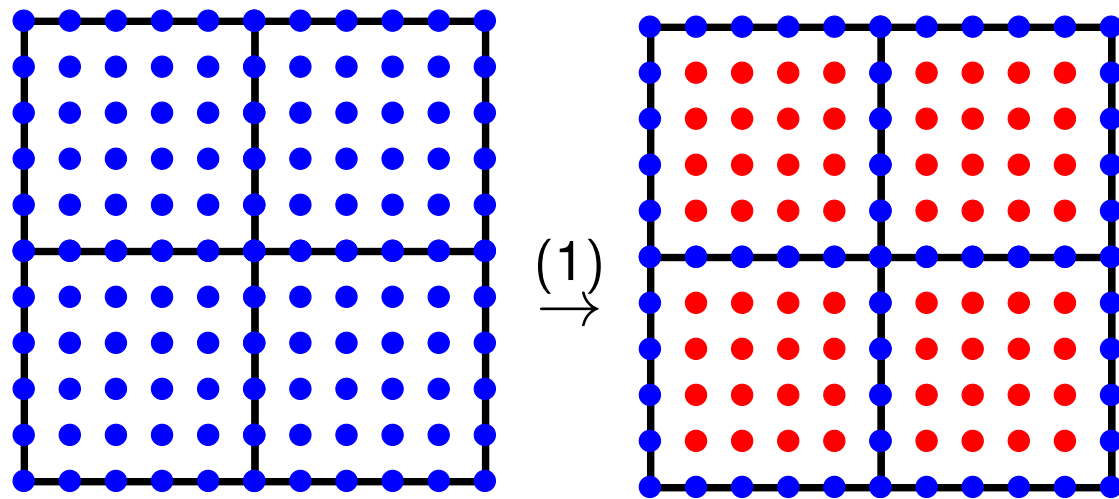
Split the domain into “small” patches we call “leaves” (they will be organized in a tree).



The original grid.

Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition. Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square. Split the domain into “small” patches we call “leaves” (they will be organized in a tree). On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”)

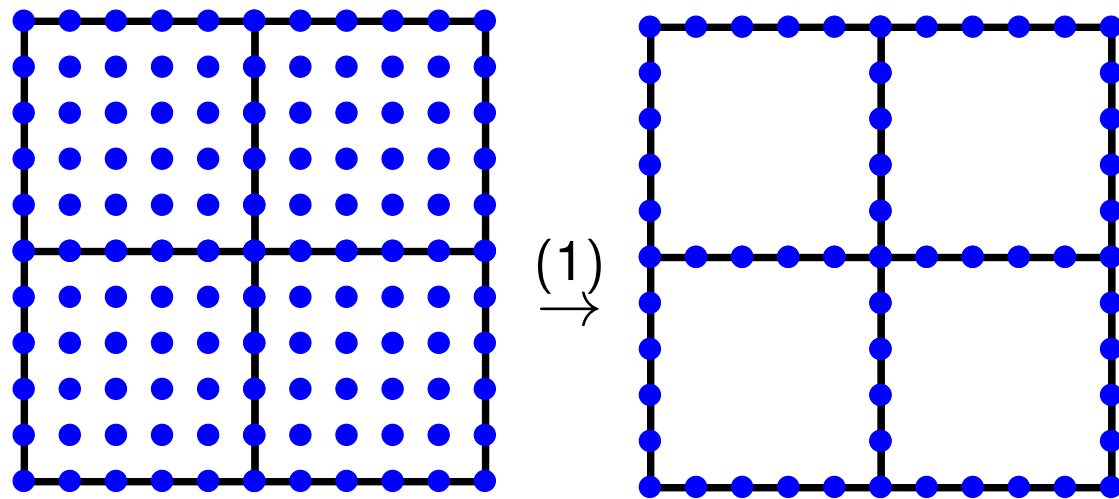


The original grid.

Leaves reduced.

Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition. Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square. Split the domain into “small” patches we call “leaves” (they will be organized in a tree). On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”)



The original grid.

Leaves reduced.

Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition.

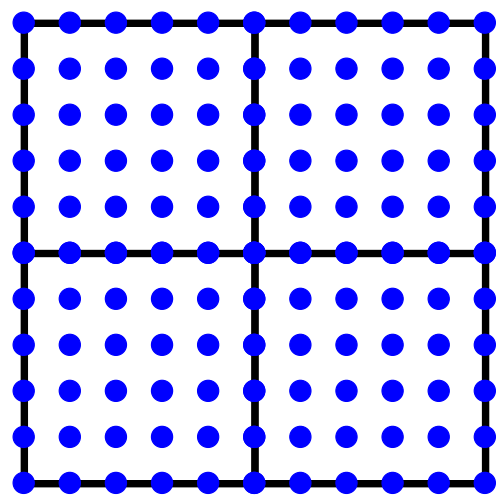
Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator).

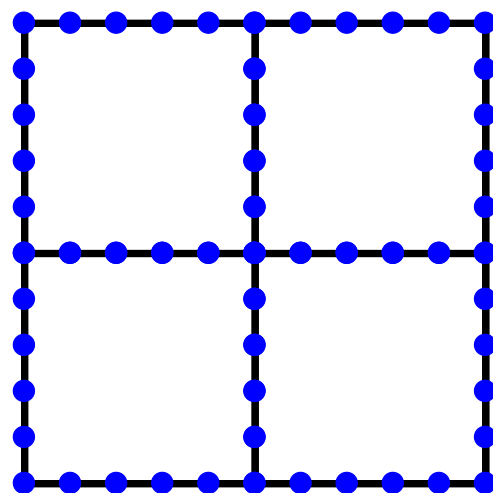
This eliminates “internal” grid points from the computation. (“Static condensation.”)

Merge the leaves in pairs of two.



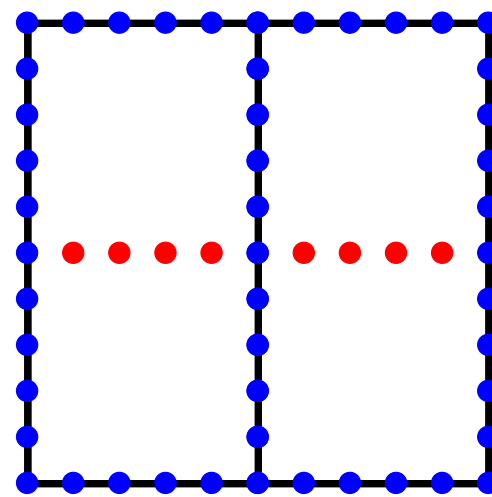
The original grid.

(1)
→



Leaves reduced.

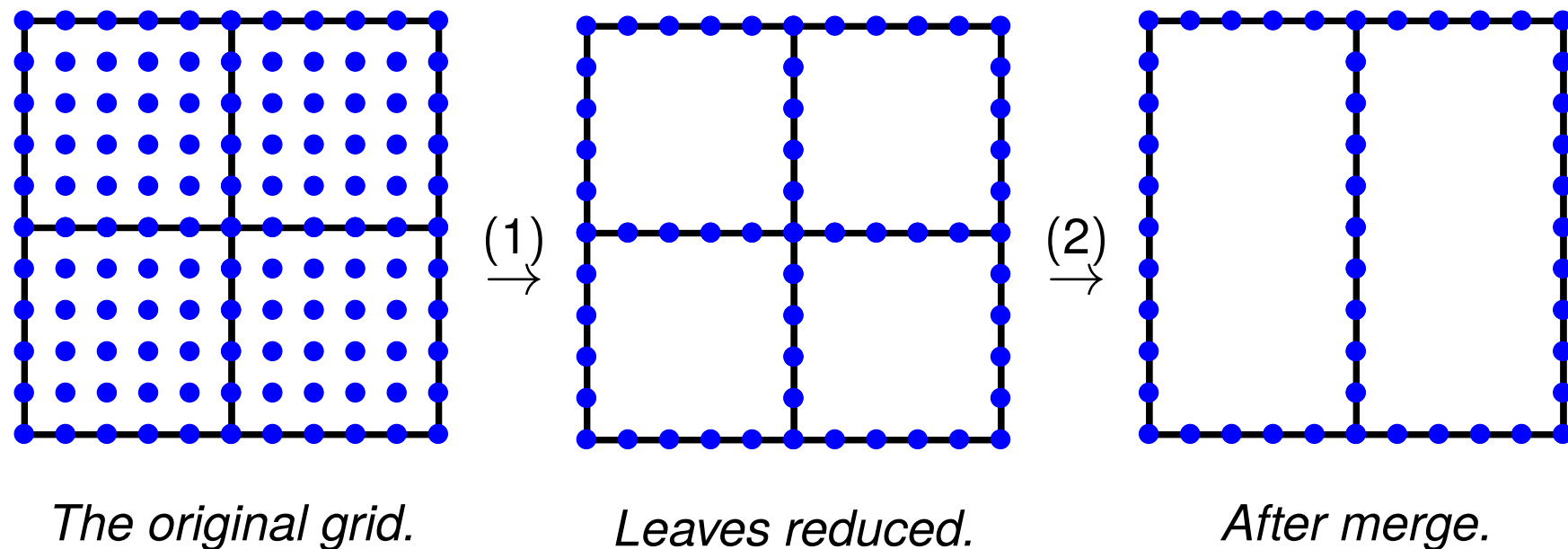
(2)
→



After merge.

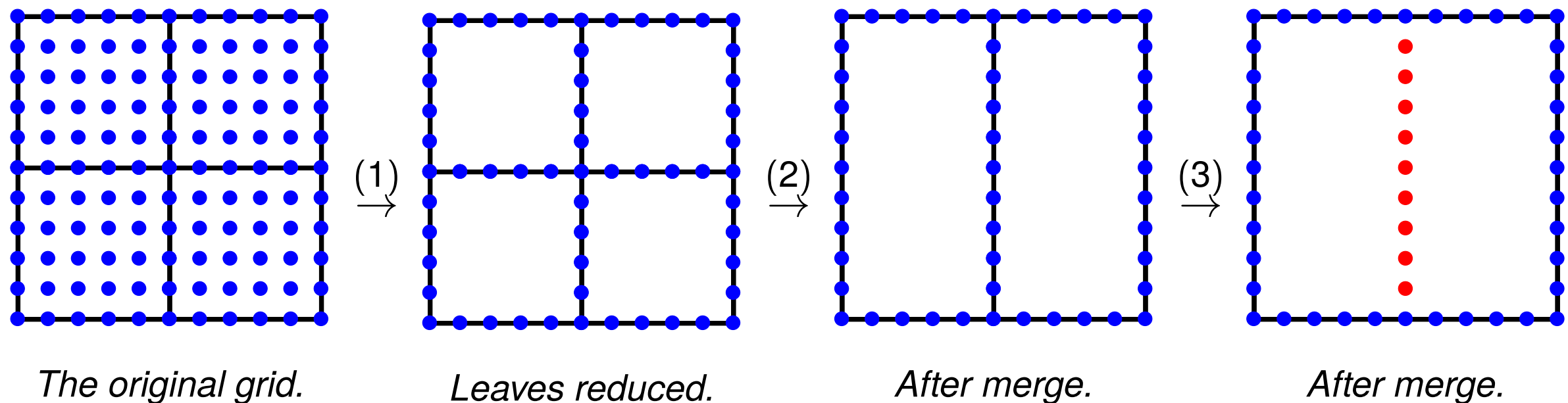
Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition. Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square. Split the domain into “small” patches we call “leaves” (they will be organized in a tree). On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”) Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves.



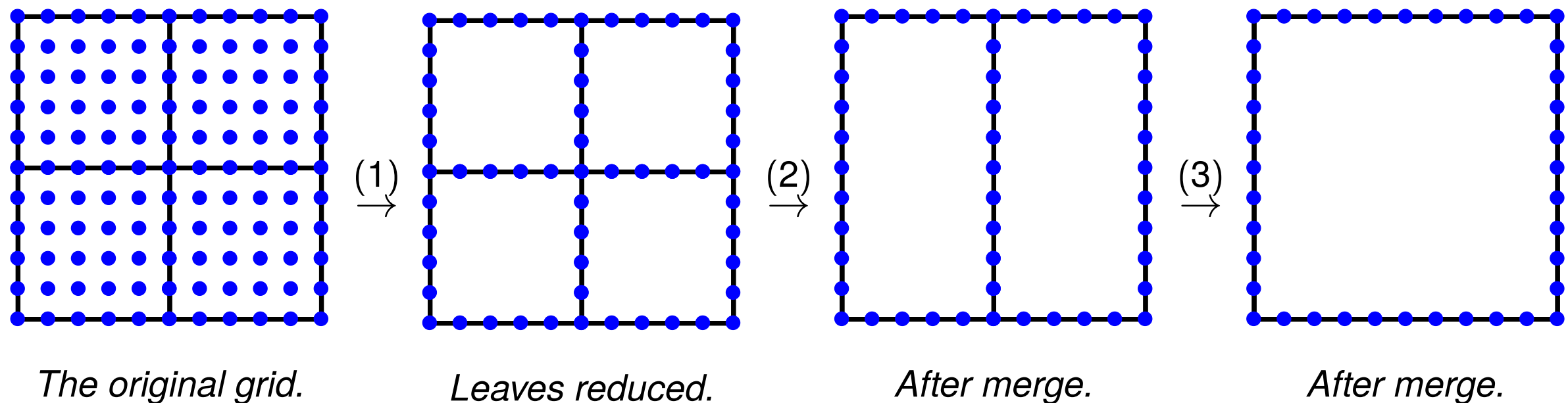
Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition. Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square. Split the domain into “small” patches we call “leaves” (they will be organized in a tree). On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”) Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves. Continue merging by pairs, organizing the domain in a tree of patches.



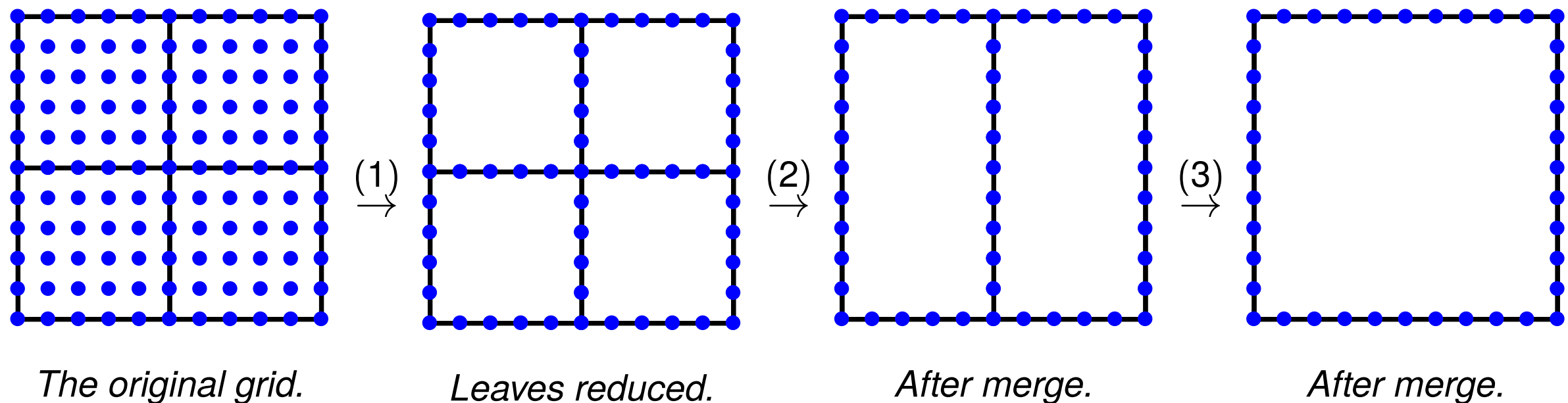
Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition. Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square. Split the domain into “small” patches we call “leaves” (they will be organized in a tree). On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”) Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves. Continue merging by pairs, organizing the domain in a tree of patches.



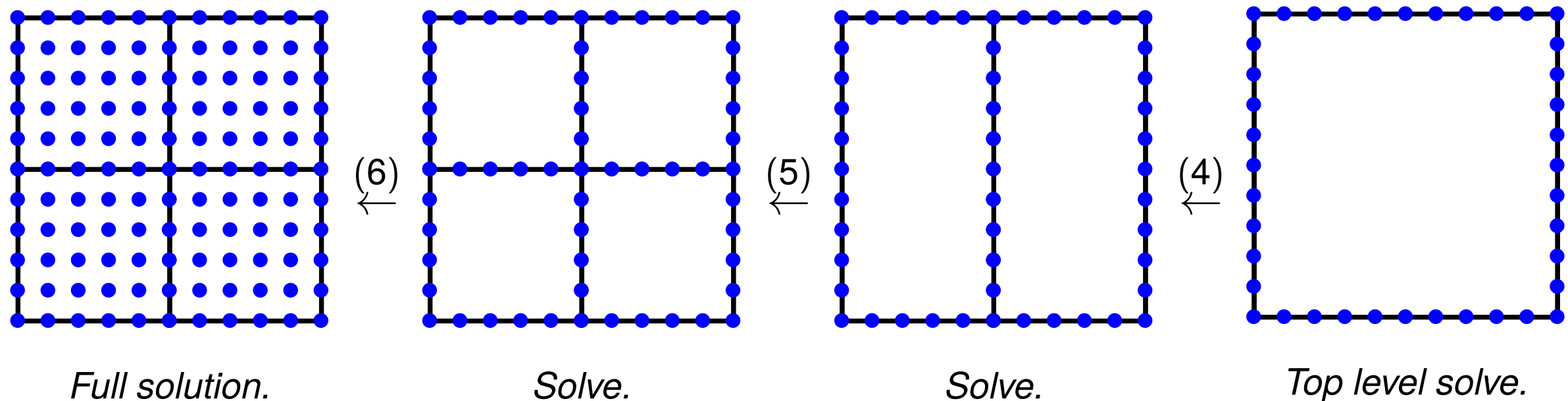
Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition. Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square. Split the domain into “small” patches we call “leaves” (they will be organized in a tree). On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”) Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves. Continue merging by pairs, organizing the domain in a tree of patches. When you reach the top level, perform a solve on the reduced problem by brute force.



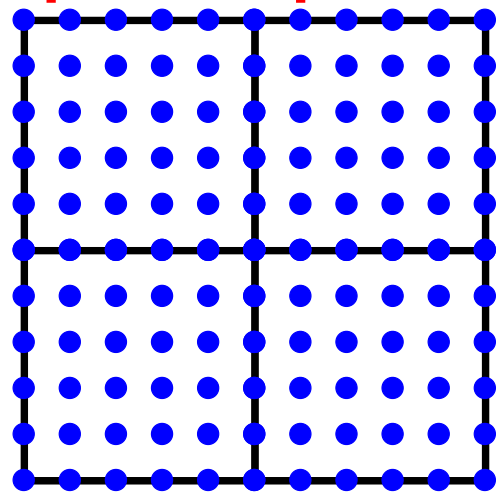
Connection between BIE and sparse direct solvers

All direct solvers to be described are based on hierarchical domain decomposition. Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square. Split the domain into “small” patches we call “leaves” (they will be organized in a tree). On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”) Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves. Continue merging by pairs, organizing the domain in a tree of patches. When you reach the top level, perform a solve on the reduced problem by brute force. Then reconstruct the solution at all internal points via a downwards pass.



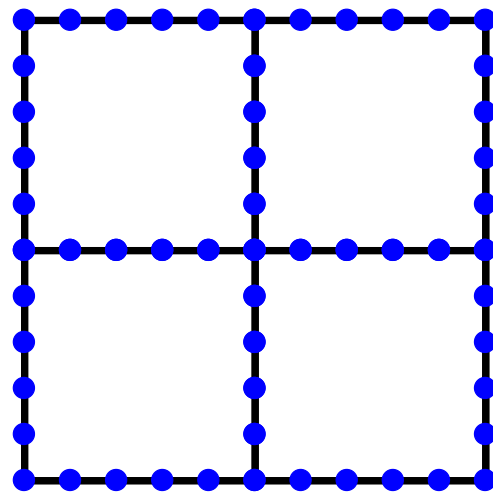
Connection between BIE and sparse direct solvers

Upwards pass — build all solution operators:



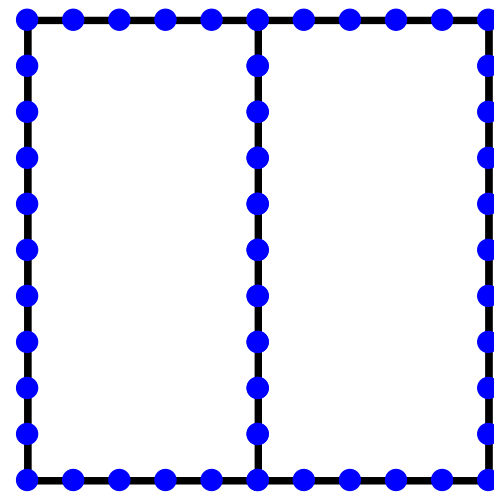
The original grid.

(1)
→



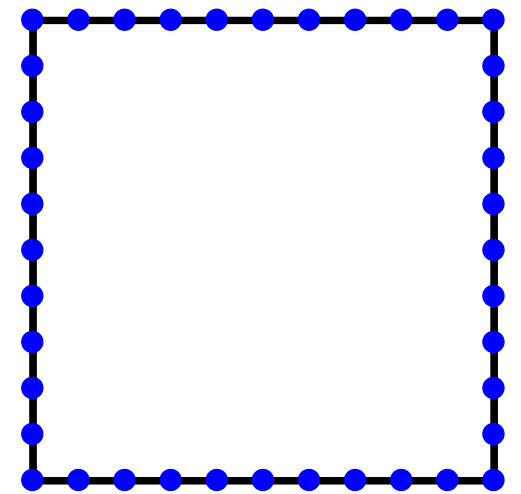
Leaves reduced.

(2)
→



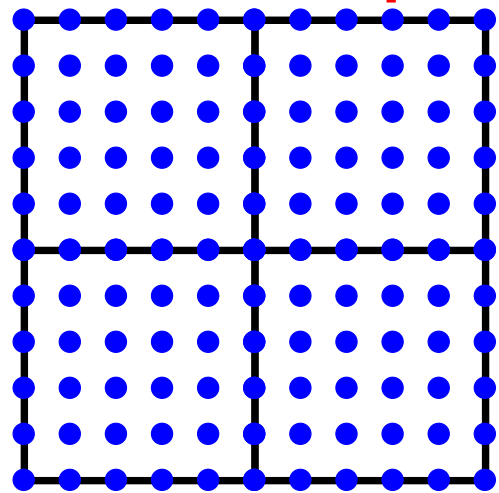
After merge.

(3)
→



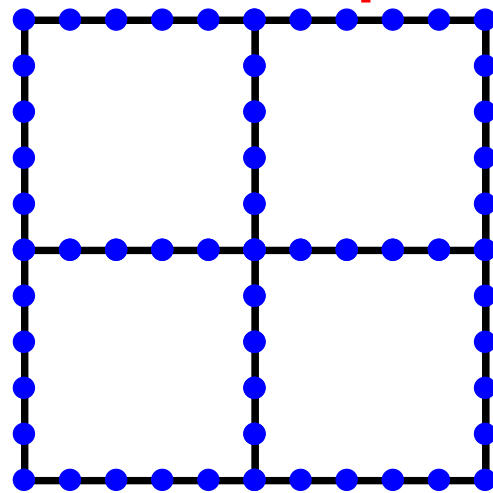
After merge.

Downwards pass — solve for a particular data function (very fast!):



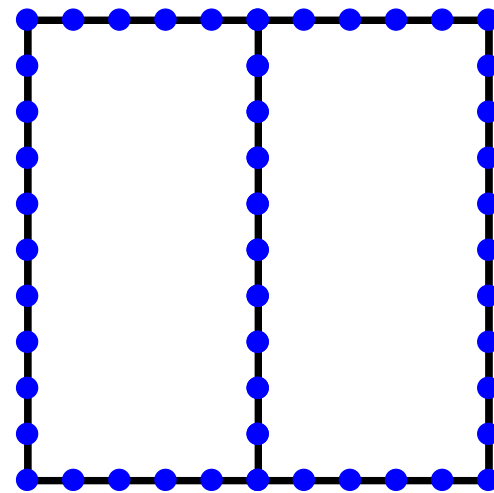
Full solution.

(6)
←



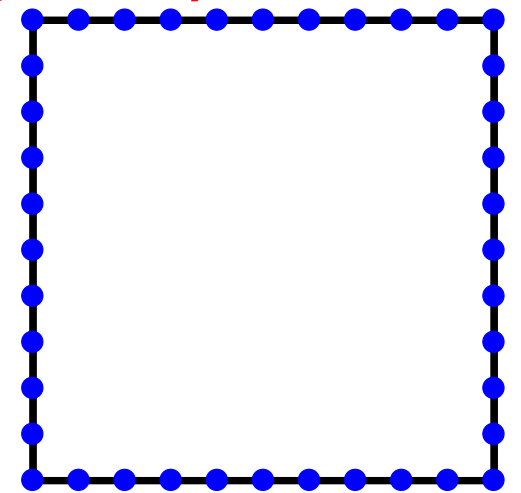
Solve.

(5)
←



Solve.

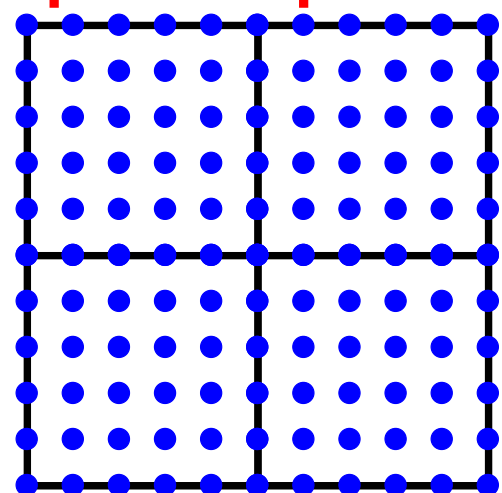
(4)
←



Top level solve.

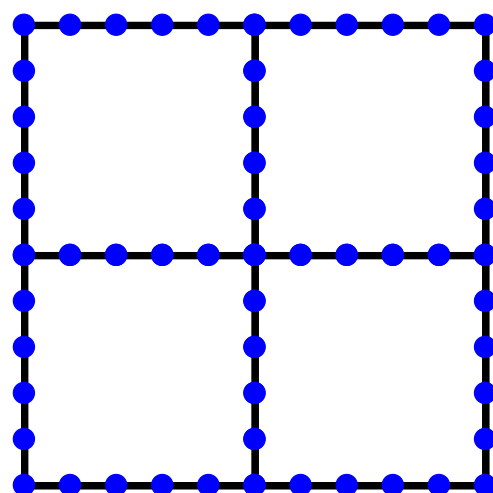
Connection between BIE and sparse direct solvers

Upwards pass — build all solution operators:



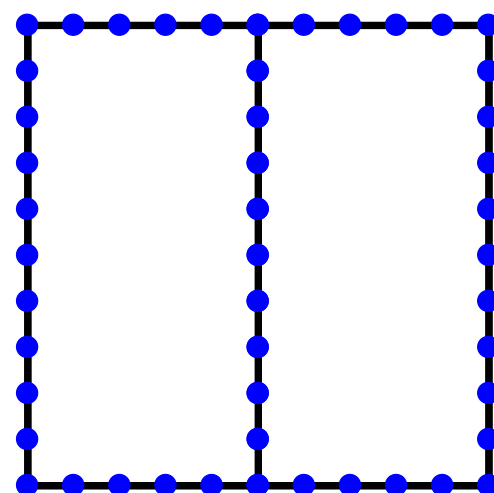
The original grid.

(1)
→



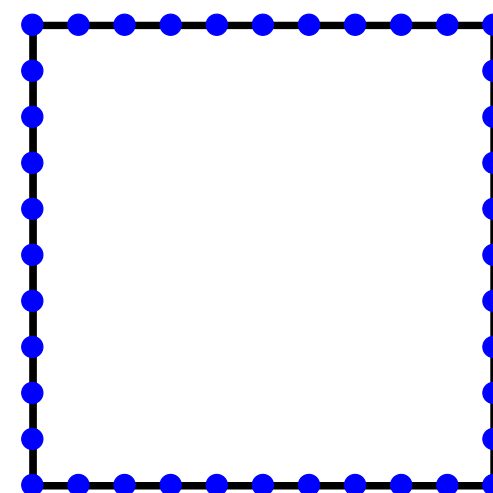
Leaves reduced.

(2)
→



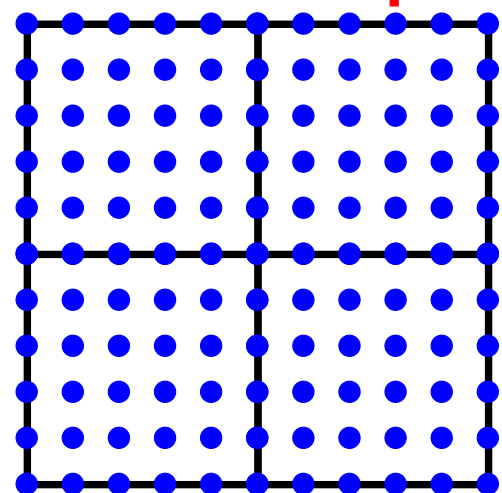
After merge.

(3)
→



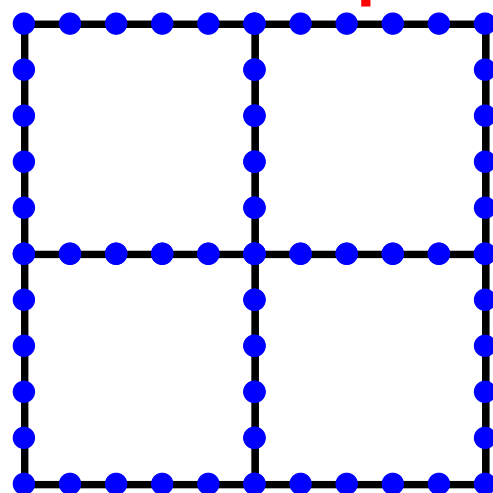
After merge.

Downwards pass — solve for a particular data function (very fast!):



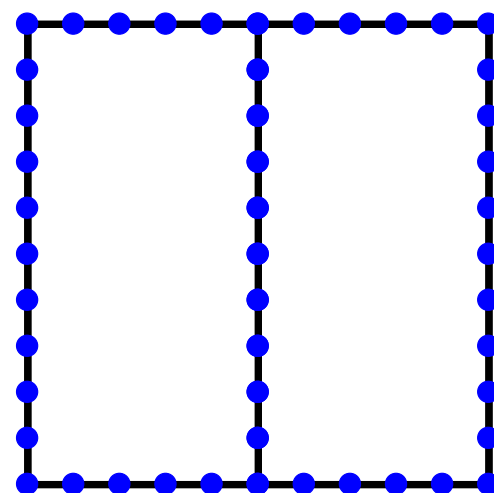
Full solution.

(6)
←



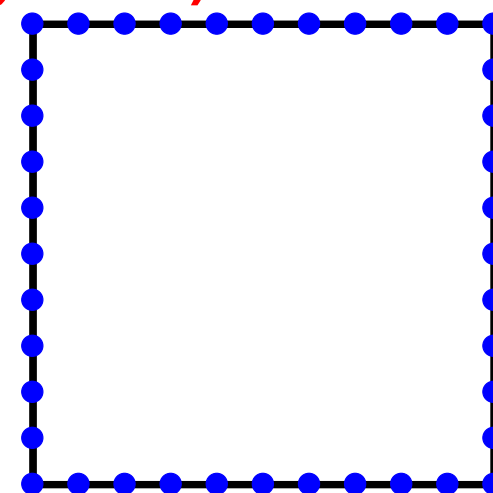
Solve.

(5)
←



Solve.

(4)
←

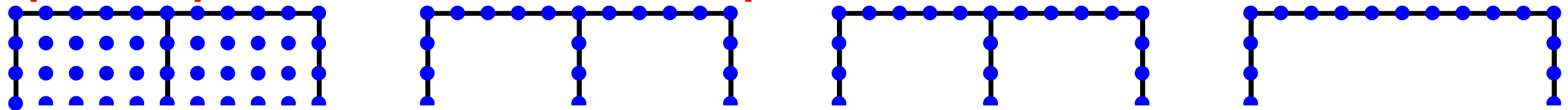


Top level solve.

Well-established idea: Classical multifrontal / nested dissection method (1973).

Connection between BIE and sparse direct solvers

Upwards pass — build all solution operators:



D



Alan George



Iain Duff

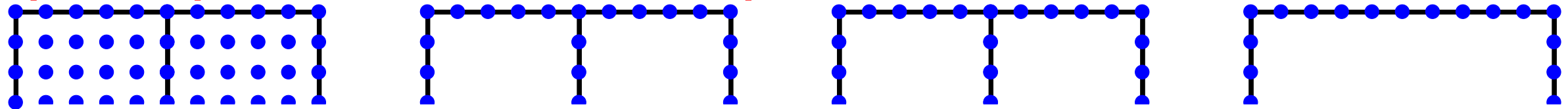


Tim Davis

Well-established idea: Classical multifrontal / nested dissection method (1973).

Connection between BIE and sparse direct solvers

Upwards pass — build all solution operators:



D



Alan George



Iain Duff



Tim Davis

Well-established idea: Classical multifrontal / nested dissection method (1973).

More recent idea: Extend this algorithmic template to discretized integral equations.

Connection between BIE and sparse direct solvers

The direct solver described works very well for moderate problem sizes.

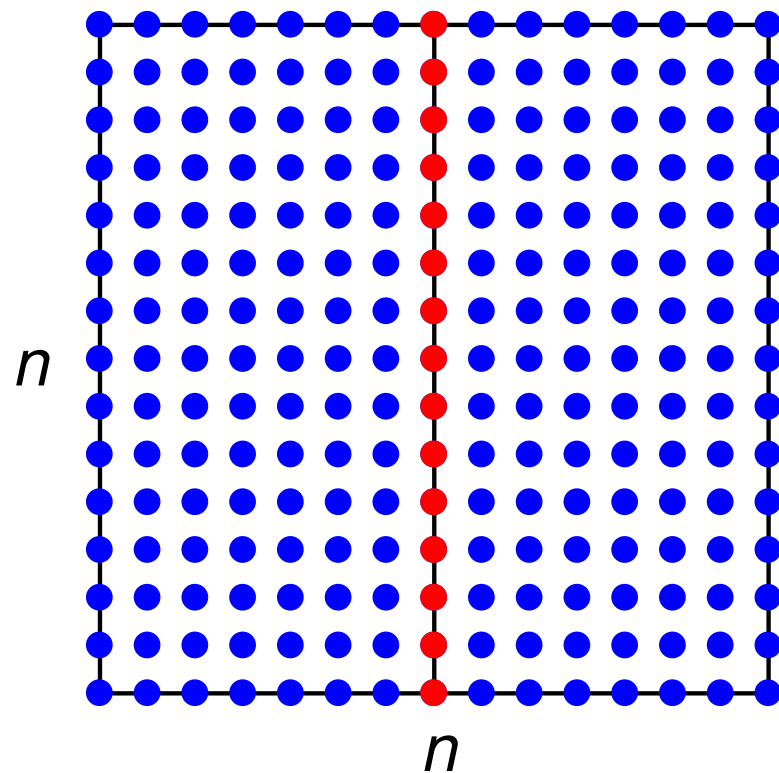
But problems arise as the number of discretization points increases . . .

Connection between BIE and sparse direct solvers

The direct solver described works very well for moderate problem sizes.

But problems arise as the number of discretization points increases ...

Consider a regular grid in 2D with $N = n \times n$ total nodes. The top level merge requires inversion of a matrix representing interactions between the red nodes:



$$N = n \times n$$

$$n = N^{1/2}$$

Since this dense matrix is of size $n \times n$, the cost for the merge is

$$\text{COST} \sim n^3 \sim (N^{1/2})^3 \sim N^{3/2}.$$

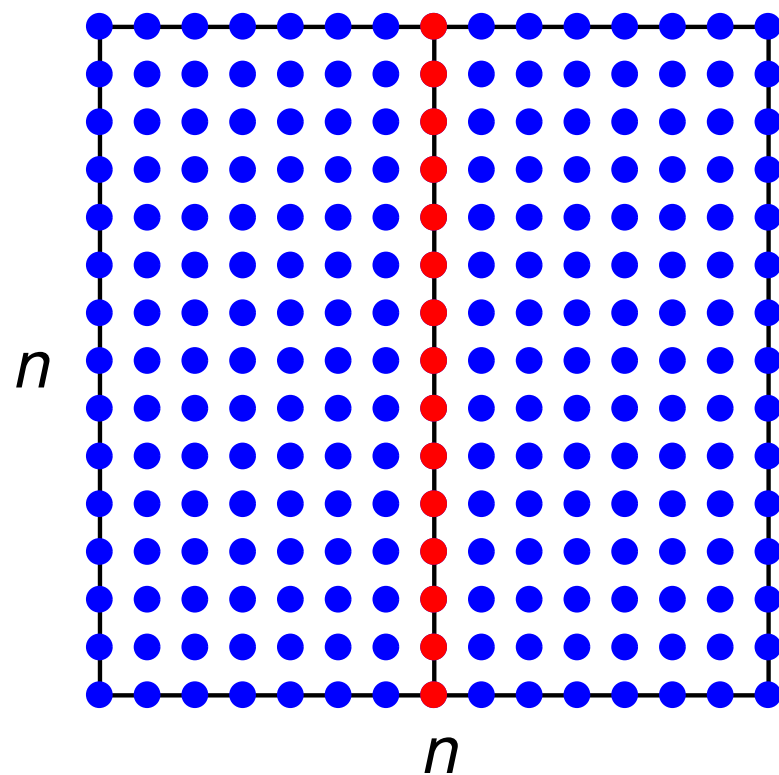
Good news: The $N^{3/2}$ term does not “kick in” until N is huge! Say $N \sim 10^8$.

Connection between BIE and sparse direct solvers

The direct solver described works very well for moderate problem sizes.

But problems arise as the number of discretization points increases ...

Consider a regular grid in 2D with $N = n \times n$ total nodes. The top level merge requires inversion of a matrix representing interactions between the red nodes:



$$N = n \times n$$

$$n = N^{1/2}$$

Since this dense matrix is of size $n \times n$, the cost for the merge is

$$\text{COST} \sim n^3 \sim (N^{1/2})^3 \sim N^{3/2}.$$

Good news: The $N^{3/2}$ term does not “kick in” until N is huge! Say $N \sim 10^8$.

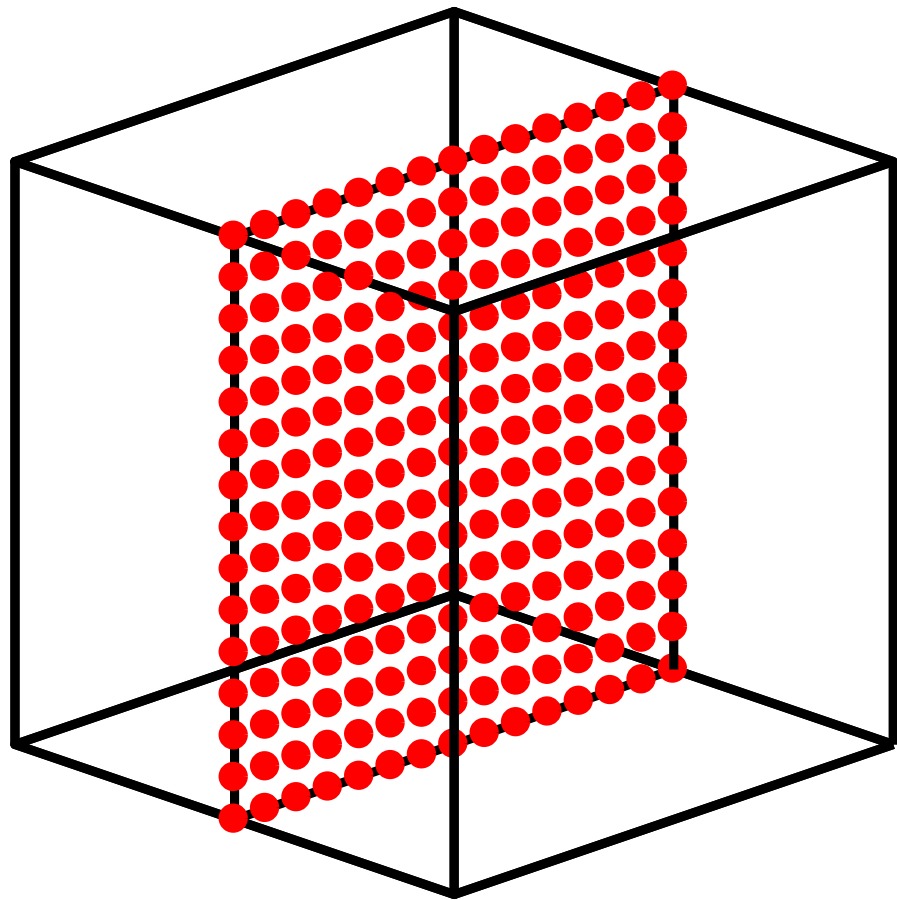
Bad news: 3D is much worse!

Connection between BIE and sparse direct solvers

The direct solver described works very well for moderate problem sizes.

But problems arise as the number of discretization points increases . . .

Consider a regular grid in 3D with $N = n \times n \times n$ total nodes. The top level merge requires inversion of a matrix representing interactions between the red nodes:



The merge requires factorization of a dense matrix of size $n^2 \times n^2$. Consequently:

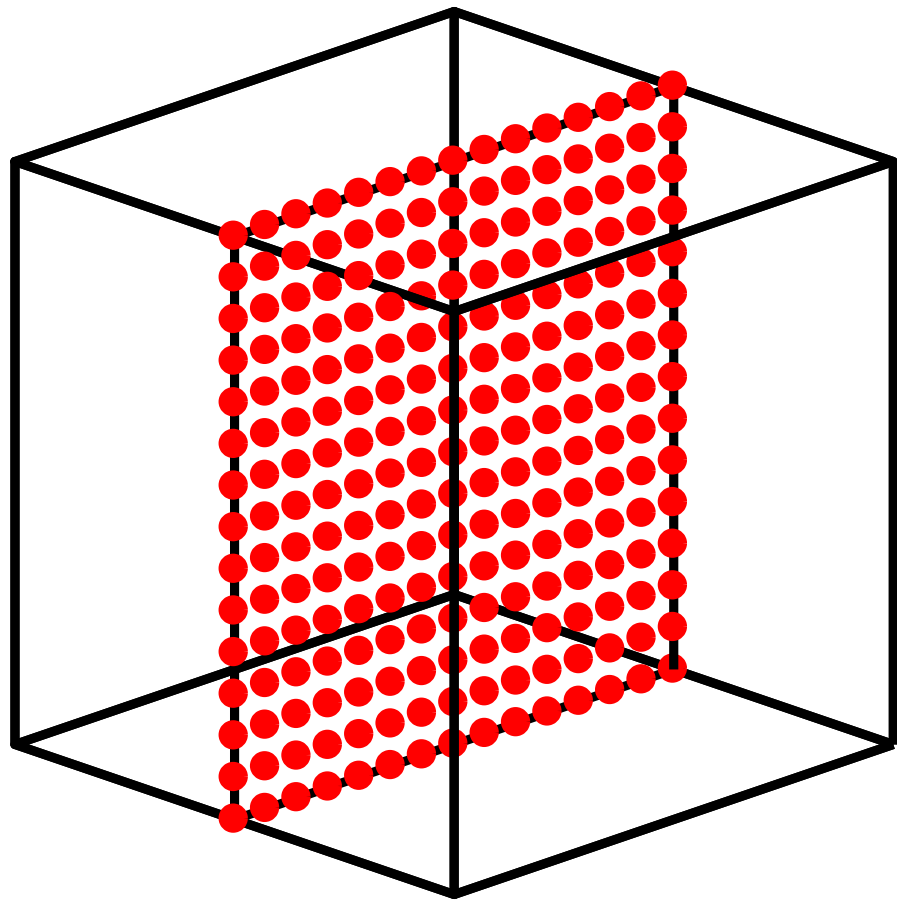
$$\text{COST} \sim (N^{1/3})^6 \sim N^2.$$

Connection between BIE and sparse direct solvers

The direct solver described works very well for moderate problem sizes.

But problems arise as the number of discretization points increases ...

Consider a regular grid in 3D with $N = n \times n \times n$ total nodes. The top level merge requires inversion of a matrix representing interactions between the red nodes:



The merge requires factorization of a dense matrix of size $n^2 \times n^2$. Consequently:

$$\text{COST} \sim (N^{1/3})^6 \sim N^2.$$

Assertion: The dense matrix very often behaves like a discretized integral operator. (E.g. Dirichlet-to-Neumann.)

It is rank-structured, and is amenable to “fast” matrix algebra.

We can reduce the complexity of the top level solve from $O(N^2)$ down to $O(N)$, and sometimes even $O(N^{2/3})$.

Connection between BIE and sparse direct solvers

Through exploiting the assertion on the previous page, the complexity of direct solvers for elliptic PDEs has in the past 10 – 20 years been decreased dramatically:

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

Connection between BIE and sparse direct solvers

Through exploiting the assertion on the previous page, the complexity of direct solvers for elliptic PDEs has in the past 10 – 20 years been decreased dramatically:

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

Key idea: Represent dense matrices using rank-structured formats (such as \mathcal{H} -matrices).

Connection between BIE and sparse direct solvers

Through exploiting the assertion on the previous page, the complexity of direct solvers for elliptic PDEs has in the past 10 – 20 years been decreased dramatically:

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

Key idea: Represent dense matrices using rank-structured formats (such as \mathcal{H} -matrices).

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.

Connection between BIE and sparse direct solvers

Through exploiting the assertion on the previous page, the complexity of direct solvers for elliptic PDEs has in the past 10 – 20 years been decreased dramatically:

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

Key idea: Represent dense matrices using rank-structured formats (such as \mathcal{H} -matrices).

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

Connection between BIE and sparse direct solvers

Key selling point: Better parallelism

Let us consider the flop counts of various parts of the computation:

	Classical Nested Dissection	Accelerated Nested Dissection
Cost to process leaves:	$\sim N$	$\sim N$
Cost to process the root:	$\sim N^2$	$\sim N^{2/3}$

Observations:

- While the dominant cost of the old scheme is processing dense matrices of size $O(N^{2/3}) \times O(N^{2/3})$, the dominant cost of the new scheme is processing the leaves.

Connection between BIE and sparse direct solvers

Key selling point: Better parallelism

Let us consider the flop counts of various parts of the computation:

	Classical Nested Dissection	Accelerated Nested Dissection
Cost to process leaves:	$\sim N$	$\sim N$
Cost to process the root:	$\sim N^2$	$\sim N^{2/3}$

Observations:

- While the dominant cost of the old scheme is processing dense matrices of size $O(N^{2/3}) \times O(N^{2/3})$, the dominant cost of the new scheme is processing the leaves.
- *The leaf computations are very easy to parallelize!*
- Parallel implementations of structured matrix algebra requires hard work (J. Poulson's dissertation; S. Li at LBNL; G. Biros; R. Kriemann; P. Amestoy, A. Buttari & T. Mary; G. Turkiyyah & D. Keyes; J. Xia; etc).
- For intermediate size problems, the structured matrices of size $O(N^{2/3}) \times O(N^{2/3})$ often fit on one machine.
- The methodology need not be all-or-nothing. Direct solvers can be used locally to handle areas with mesh refinement etc.

Connection between BIE and sparse direct solvers

Summary: The unifying theme between fast direct solvers for BIEs and fast direct solvers for PDEs is that in both cases, you reduce the dimension of the domain where the dense matrices “live” from 3D to 2D!

1. Constant coefficient problems: Reformulate as integral equation on 2D boundary.
2. Variable coefficient problems: Use a sparse direct solver as an outer solver.
This reduces the problem to dense matrices on 2D “mesh separators”.

The reduction in dimension is critically important!

Rank structured matrix algebra scales poorly with dimension!

Interaction ranks

- Why are they small?
- How small are they, exactly?

Recall that we are interested in solving the PDE
$$\begin{cases} Au(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \Omega, \\ Bu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Gamma. \end{cases} \quad (\text{BVP})$$

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

Question: Why do the dense matrices resulting upon discretization of (SLN) typically have *off-diagonal blocks of low numerical rank*?

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

This effect has many well known physical consequences:

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

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

Interaction ranks: Boundary integral equations

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

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

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

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

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

The kernels are derived from the corresponding fundamental solutions:

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

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

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

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

where, as before,

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

Interaction ranks: Boundary integral equations

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

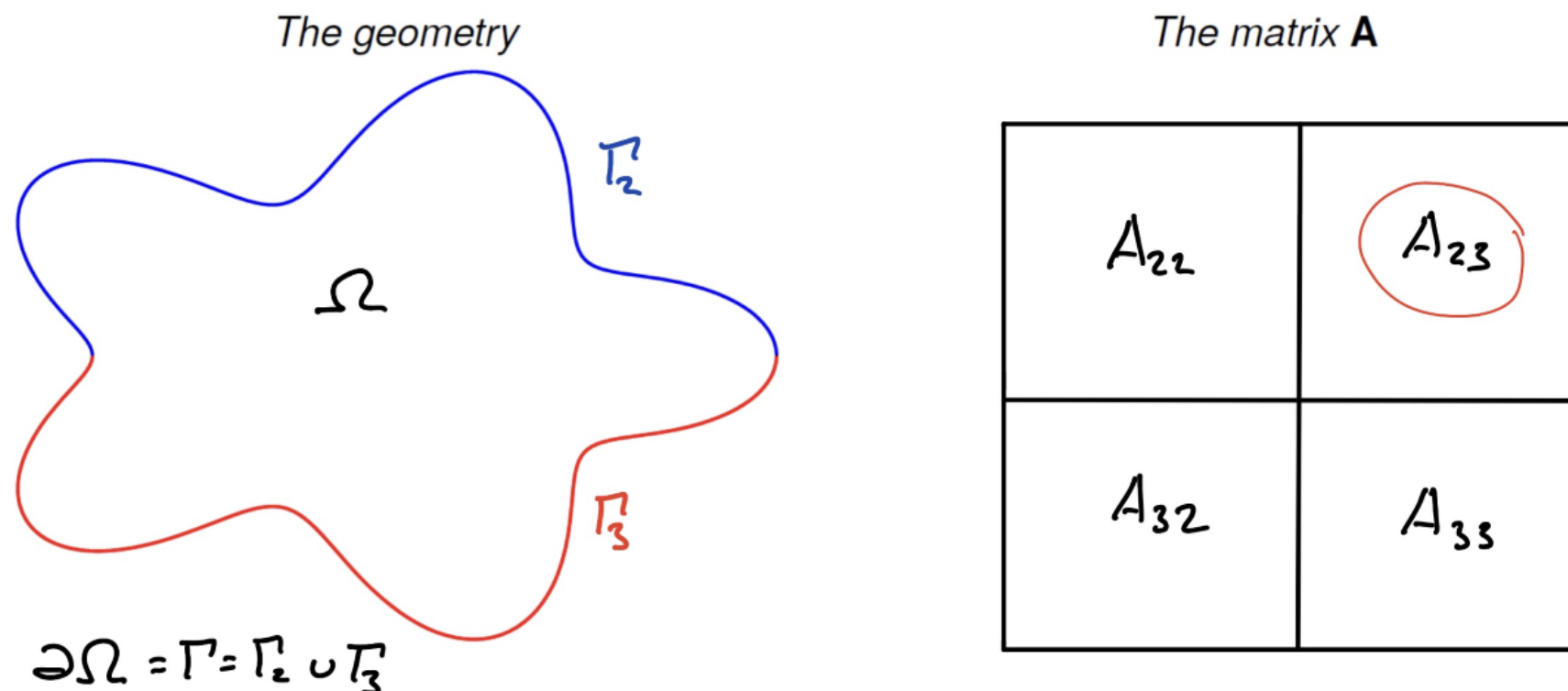
The first is a reformulation of a Dirichlet problem involving the Laplace equation:

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

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

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

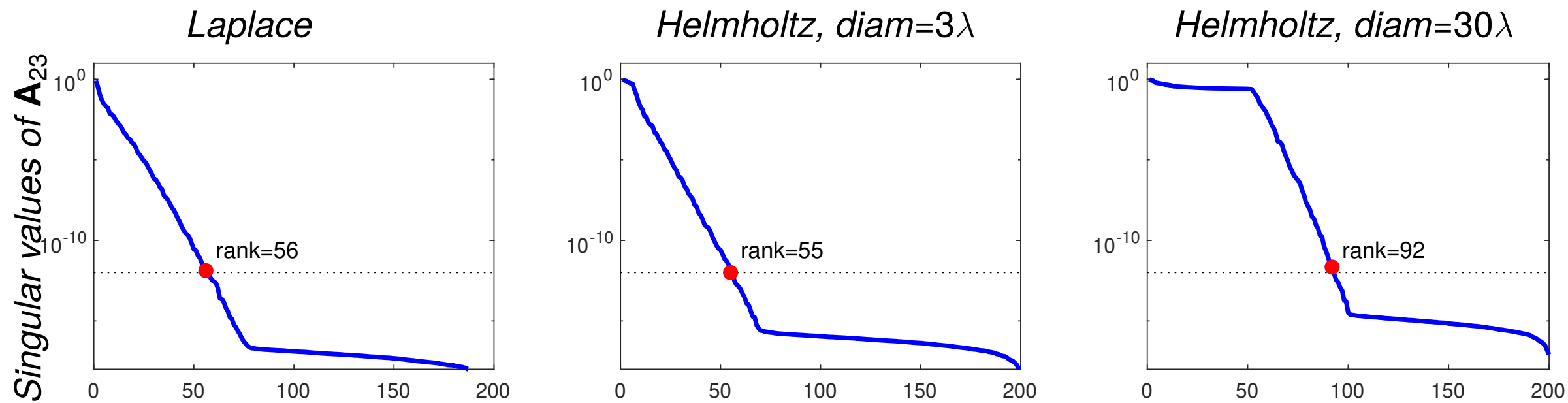
Let \mathbf{A} denote the matrix resulting from discretization of either BIE.



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

Interaction ranks: Boundary integral equations

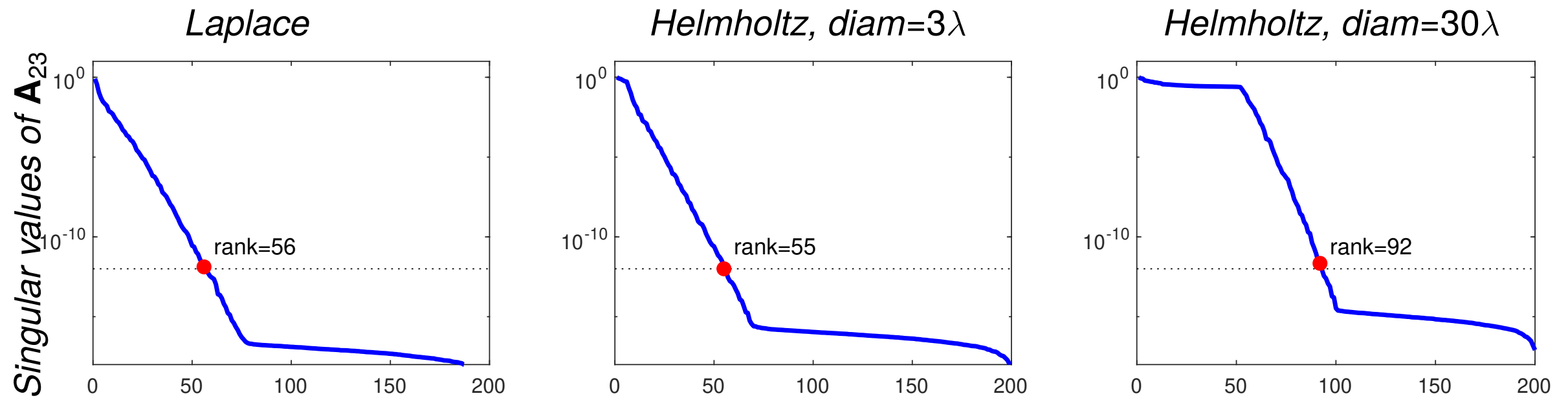
The ranks of an off-diagonal block of \mathbf{A} :



This is all as expected. Somewhat accessible by analysis.

Interaction ranks: Boundary integral equations

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

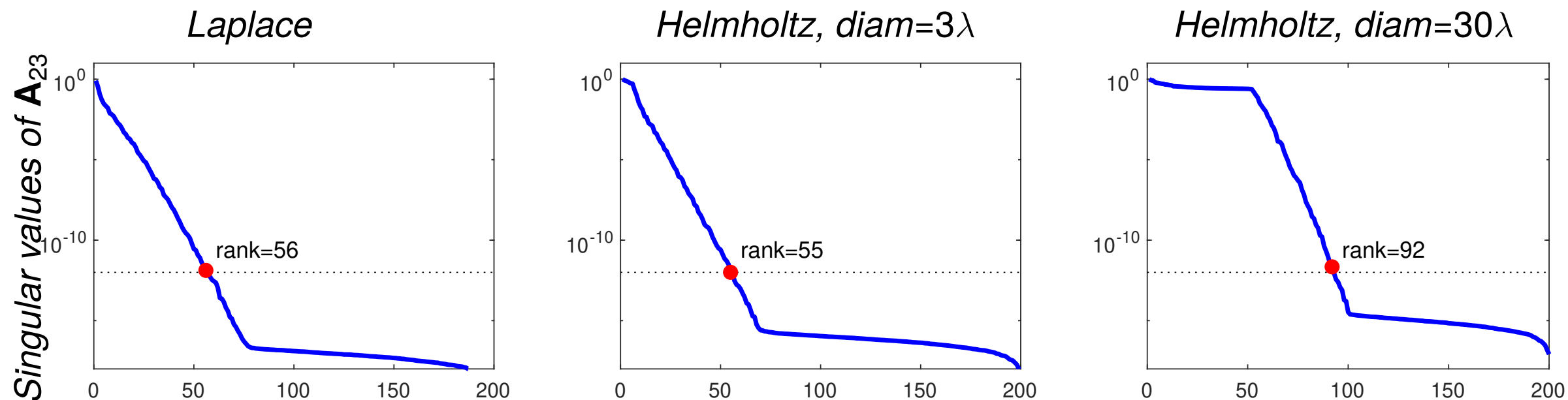


This is all as expected. Somewhat accessible by analysis.

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

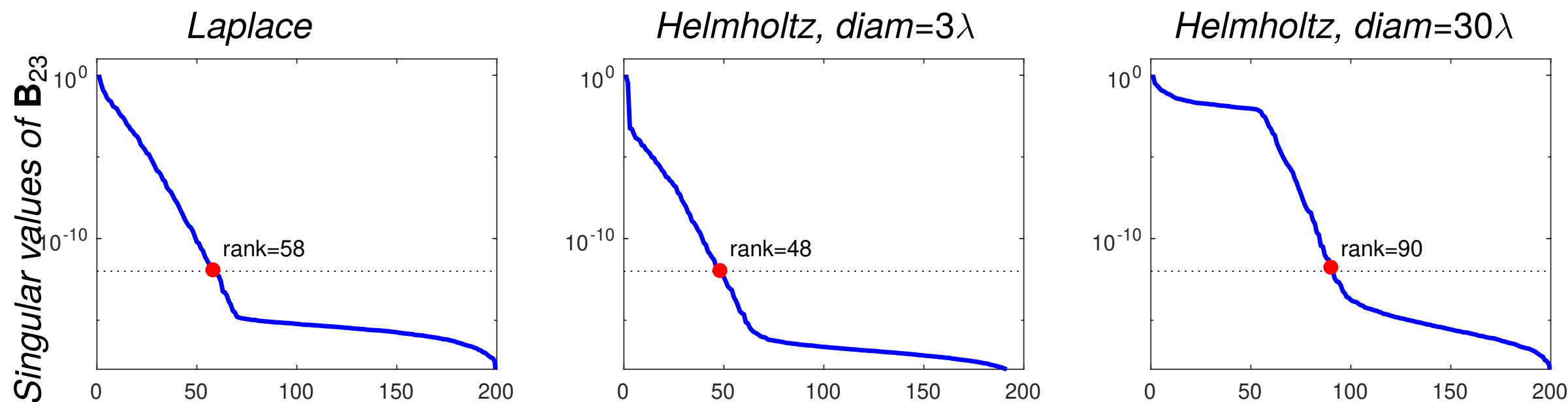
Interaction ranks: Boundary integral equations

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



This is all as expected. Somewhat accessible by analysis.

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

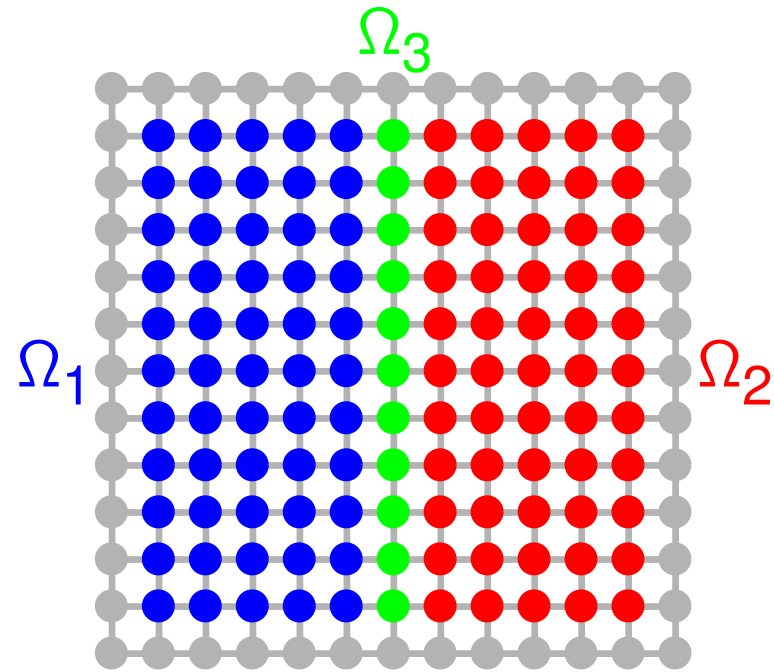


Remarkable similarity!

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

Interaction ranks: Stiffness matrix from finite difference discretization

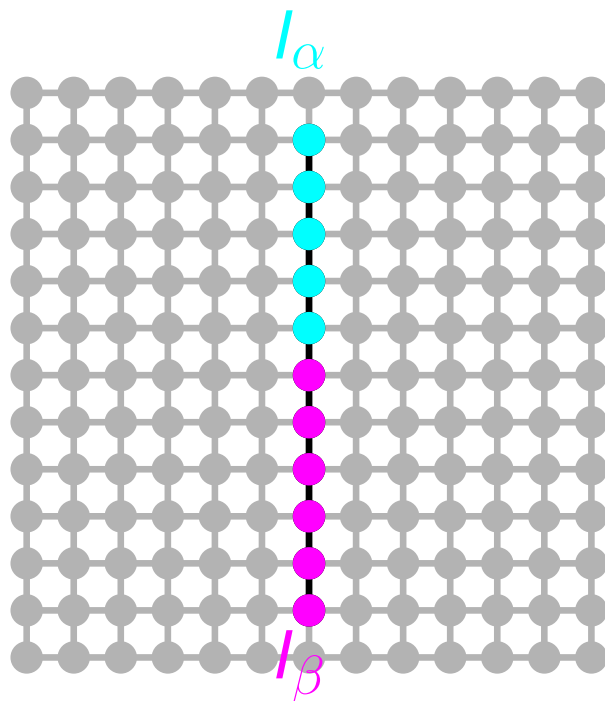
Recall our example of Laplace's equation discretized using the 5-point stencil.



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

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

Then split the Schur complement into four parts:

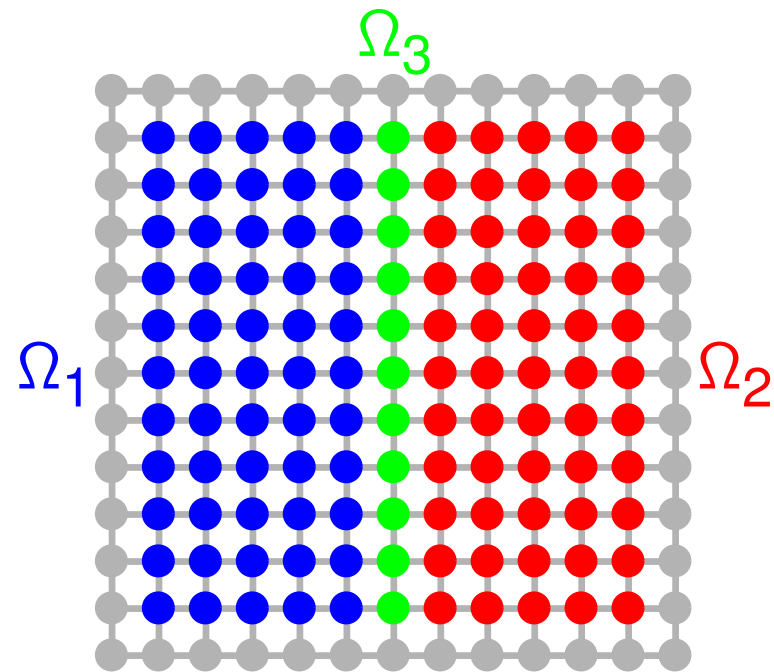


$$\mathbf{S} = \begin{array}{|c|c|} \hline \mathbf{S}_{\alpha\alpha} & \mathbf{S}_{\alpha\beta} \\ \hline \mathbf{S}_{\beta\alpha} & \mathbf{S}_{\beta\beta} \\ \hline \end{array}$$

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

Interaction ranks: Stiffness matrix from finite difference discretization

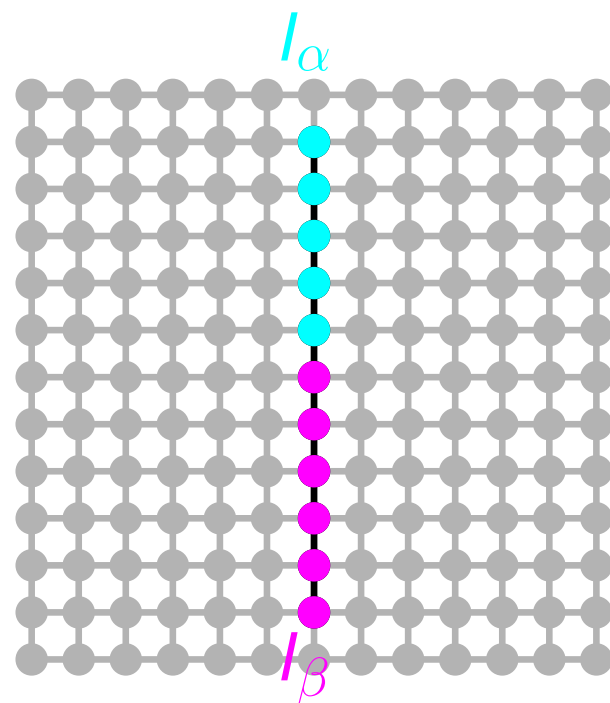
Recall our example of Laplace's equation discretized using the 5-point stencil.



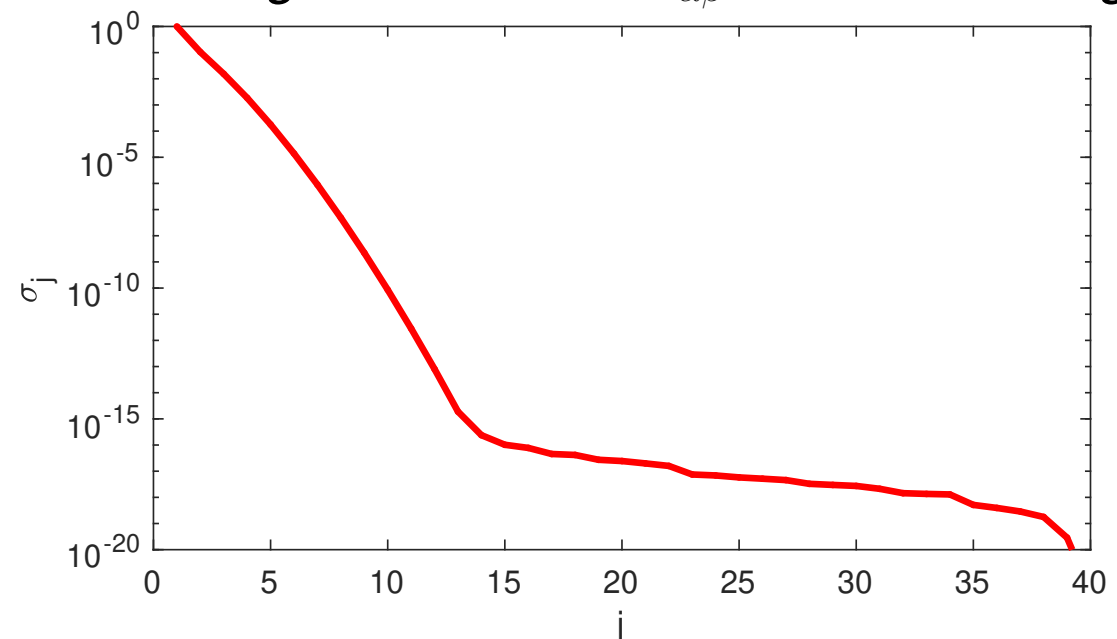
$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{A}_{13} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{bmatrix}$$

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

Then split the Schur complement into four parts:



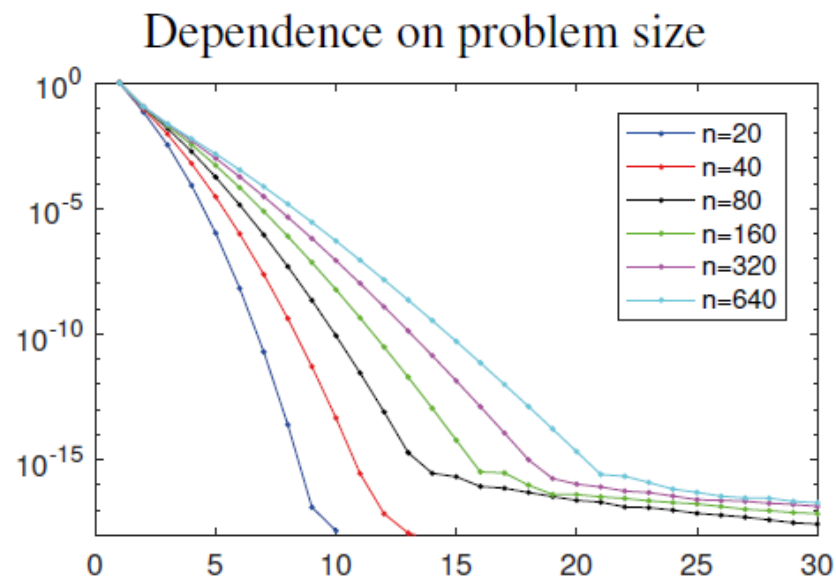
Singular values of $S_{\alpha\beta}$ for an 80×80 grid.



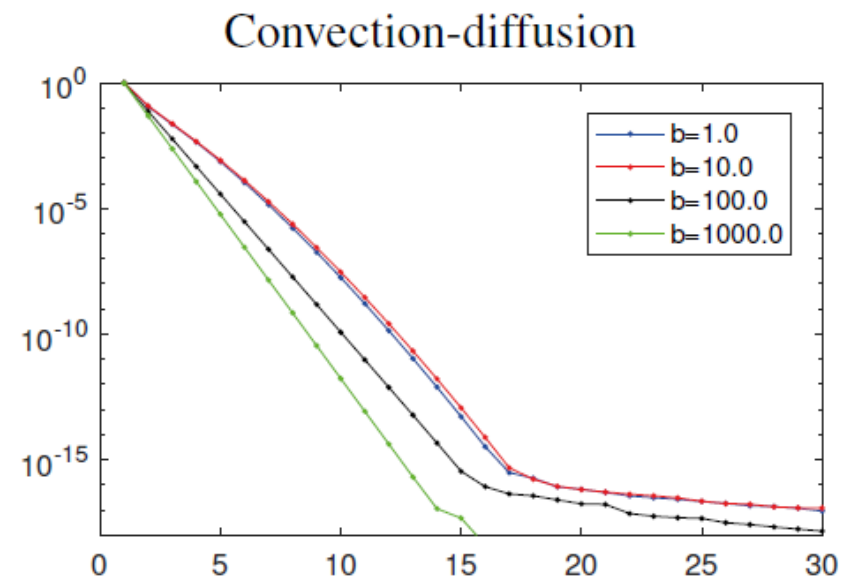
We explore the svds of $\mathbf{S}_{\alpha\beta}$ — encoding interactions between I_α and I_β .

Interaction ranks: Stiffness matrix from finite difference discretization

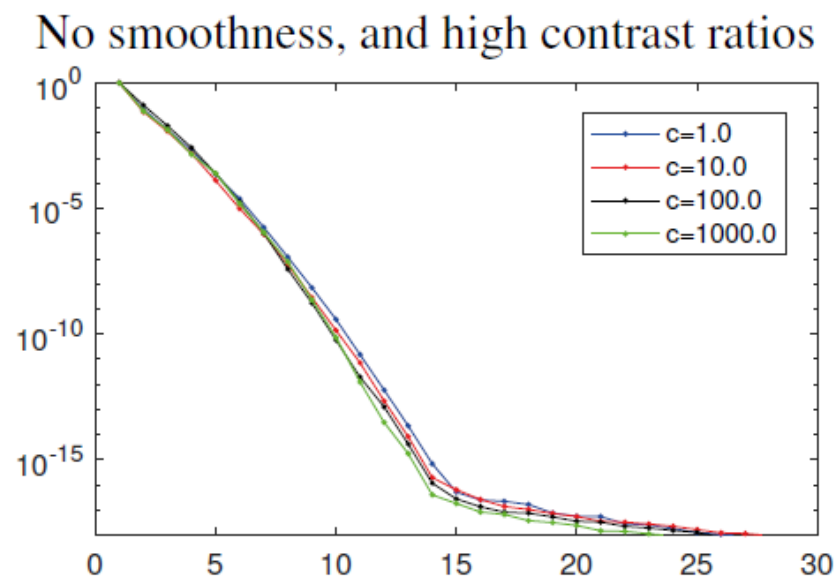
Let us try a few different PDEs, and different problem sizes:



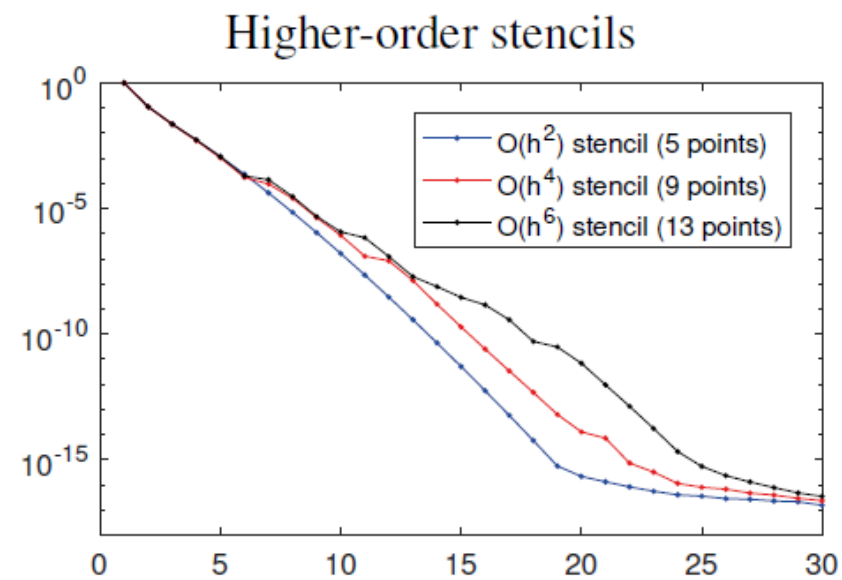
(a)



(b)



(c)



(d)

Note: The rank decay property is remarkably stable!

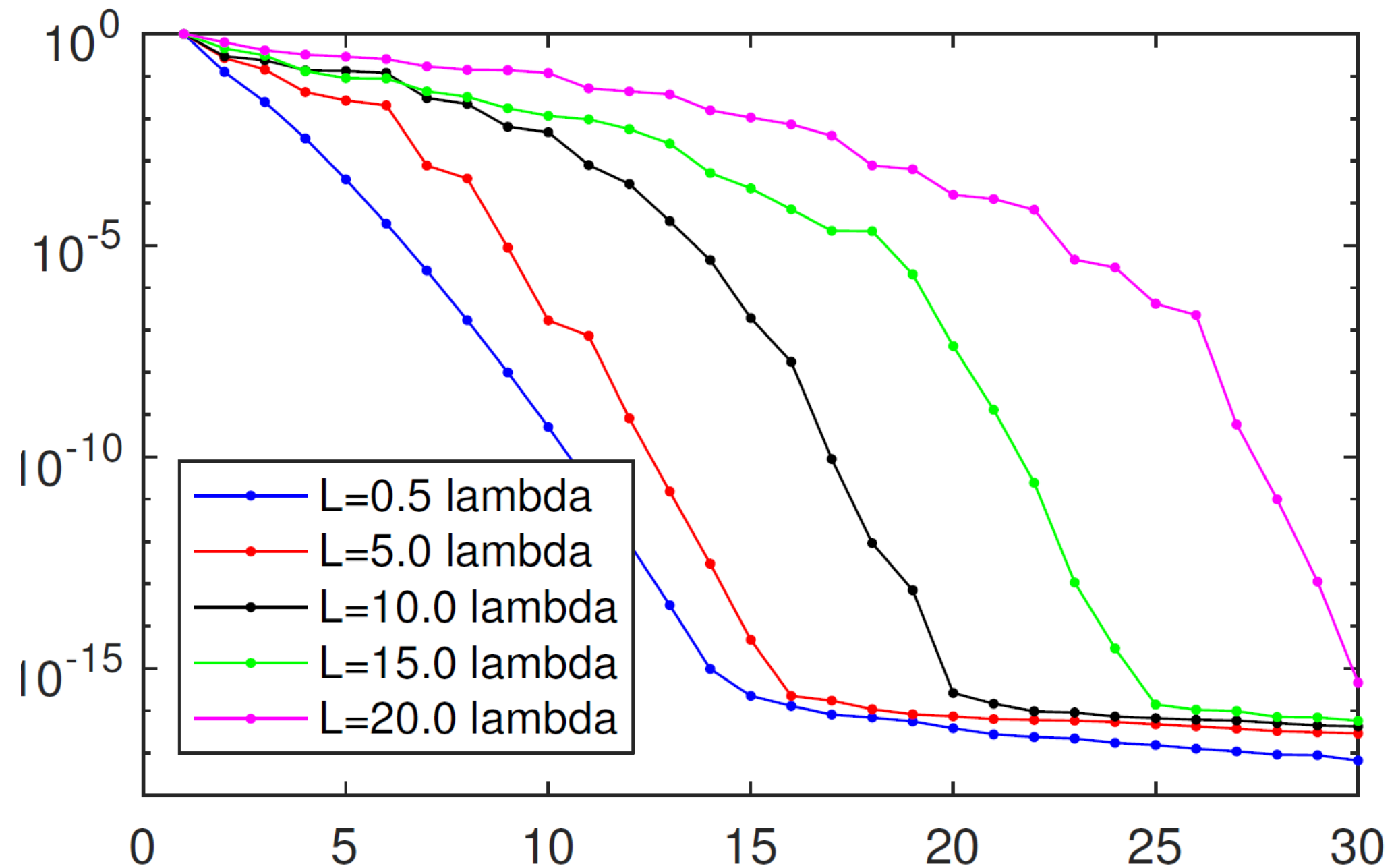
Note: The decay continues to ϵ_{mach} — regardless of the discretization errors!

Interaction ranks: Stiffness matrix from finite difference discretization

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

Interaction ranks: Stiffness matrix from finite difference discretization

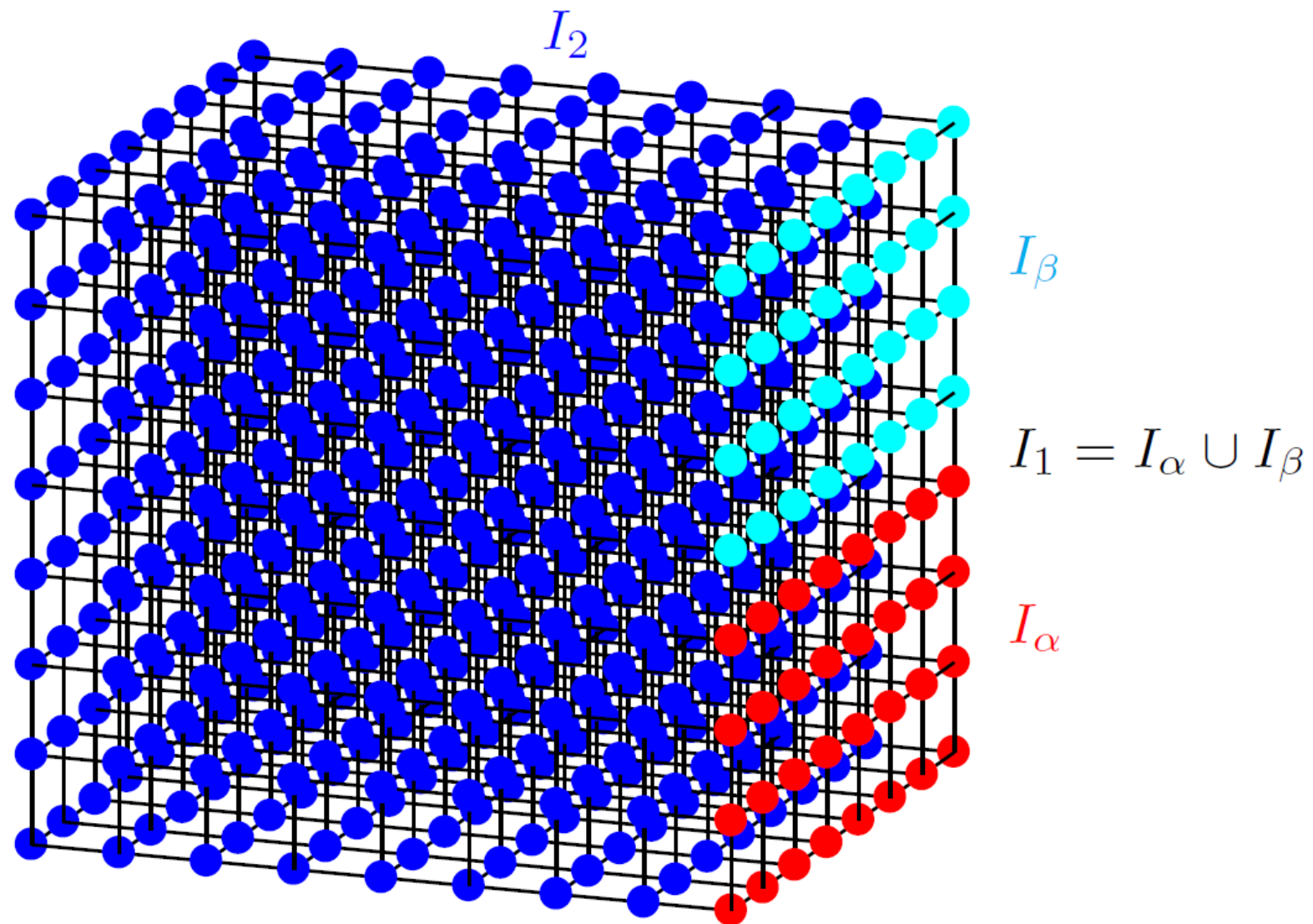
Next, let us consider Helmholtz problems with increasing wave numbers.



Fast decay *once oscillations are resolved.*

Interaction ranks: Stiffness matrix from finite difference discretization

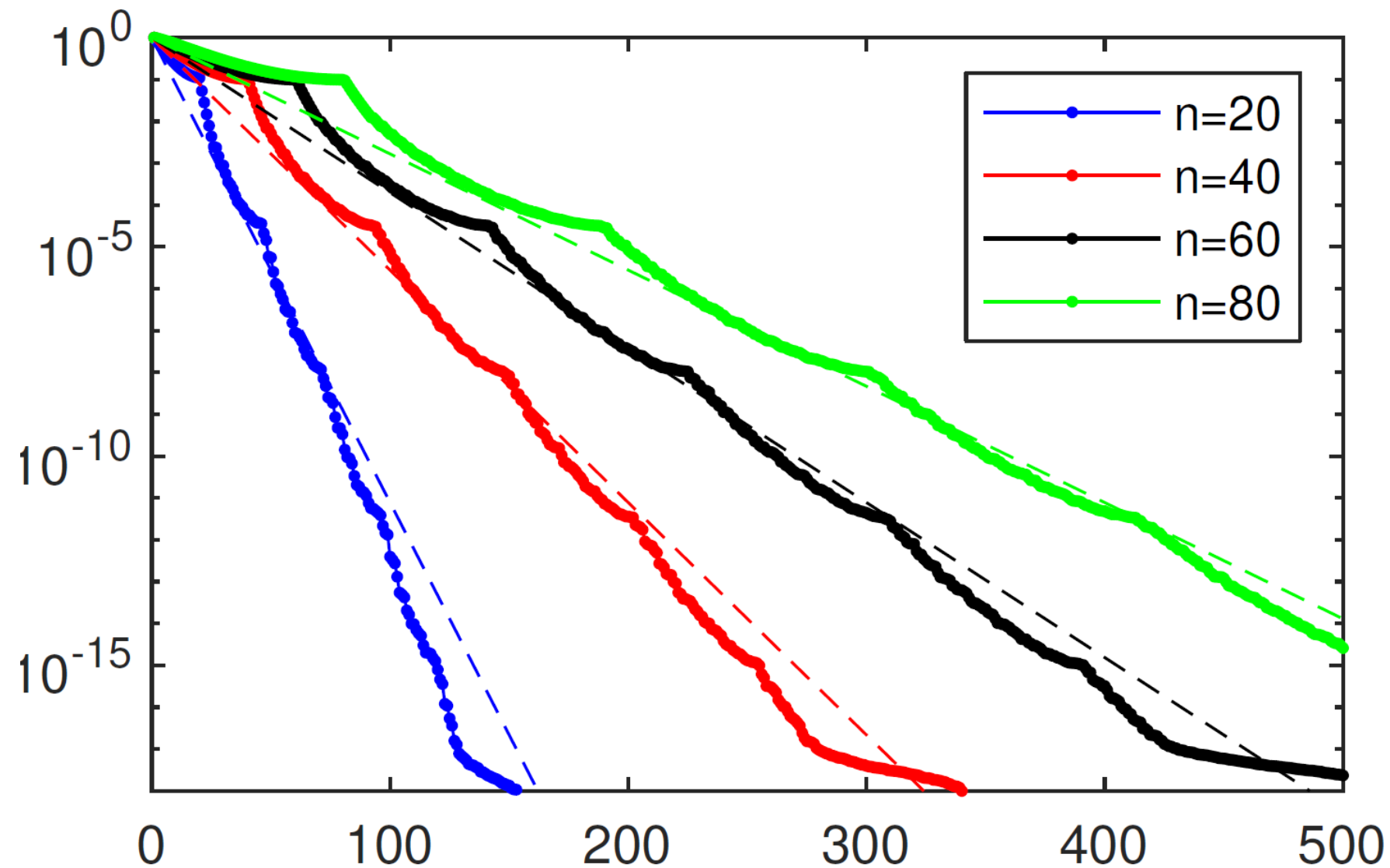
Finally, let us consider the analogous 3D problem.



The geometry.

Interaction ranks: Stiffness matrix from finite difference discretization

Finally, let us consider the analogous 3D problem.



The singular values.

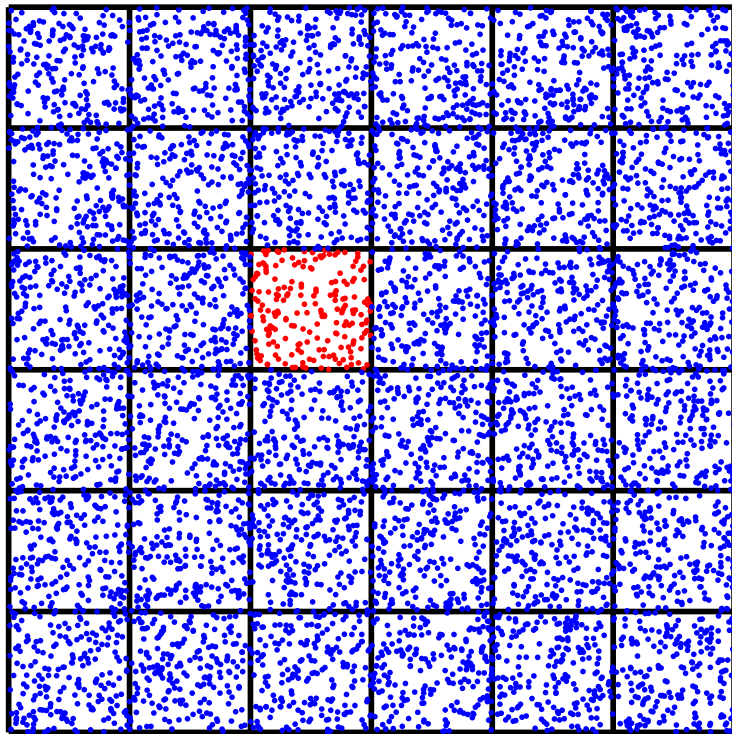
Variations of fast direct solvers

- Strong vs. weak admissibility.
- Nested basis matrices.
- Flat vs. hierarchical representations.

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

Weak admissibility: Compress directly adjacent patches.

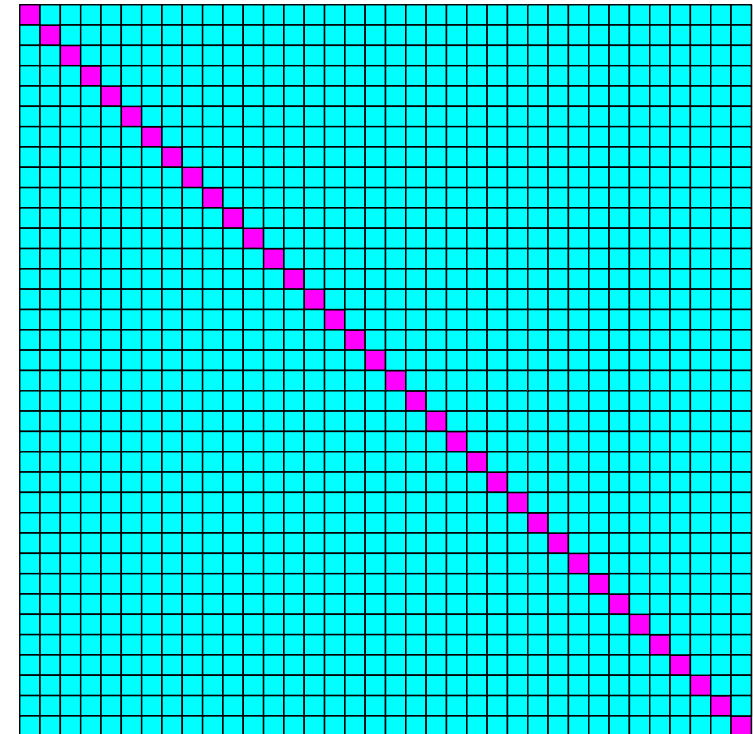
The geometry



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.

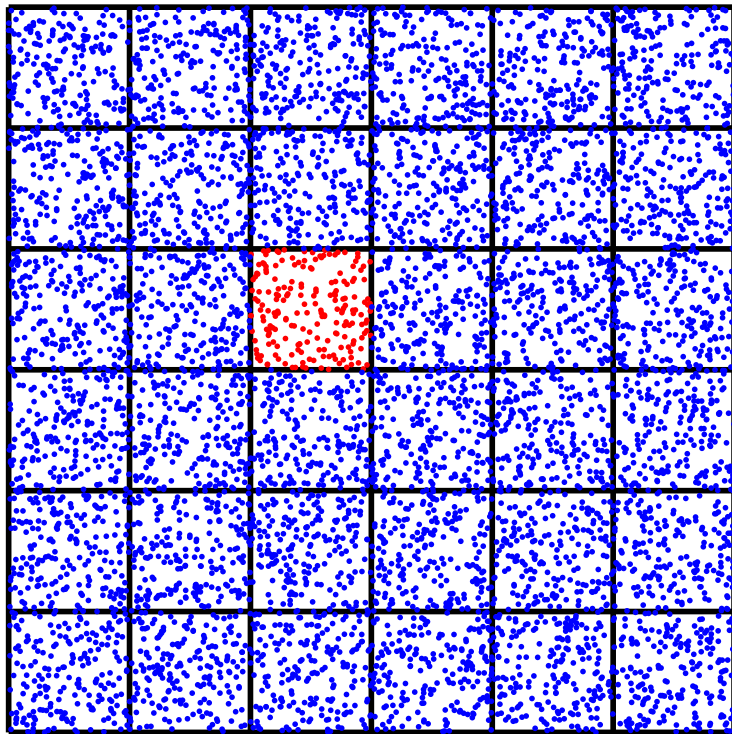
The matrix



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

Weak admissibility: Compress directly adjacent patches.

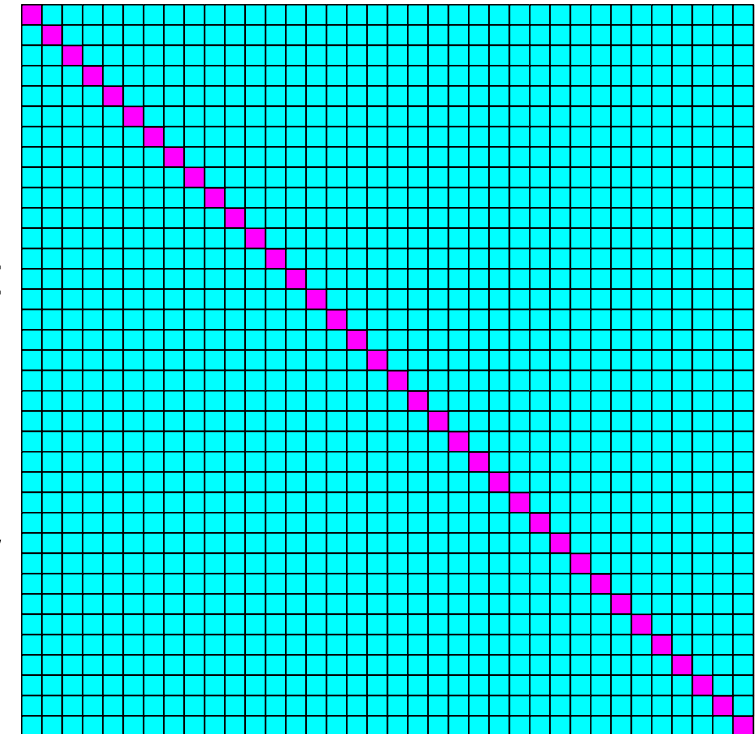
The geometry



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

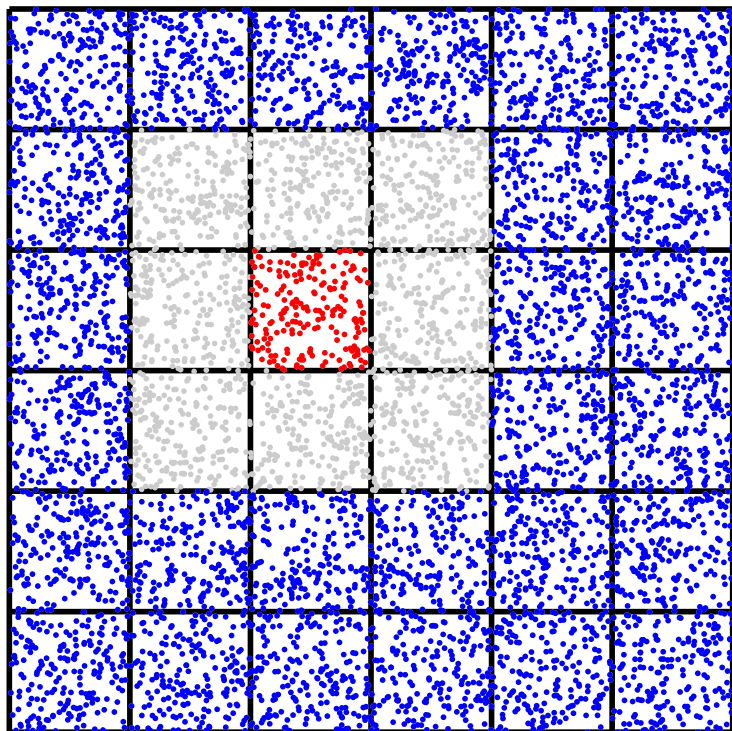
Right: Magenta blocks are dense. Cyan blocks low rank. Many low rank blocks, but high ranks.

The matrix



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

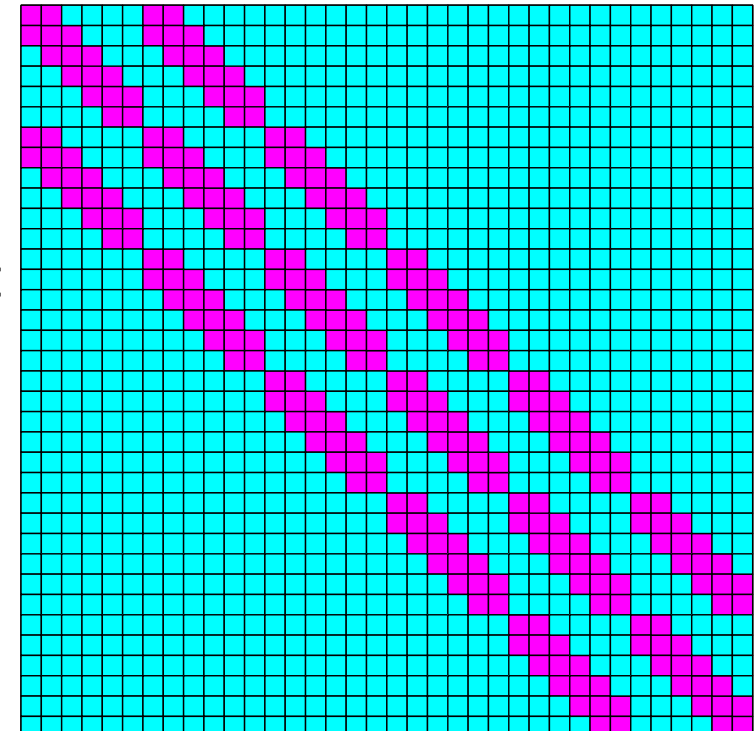
The geometry



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.

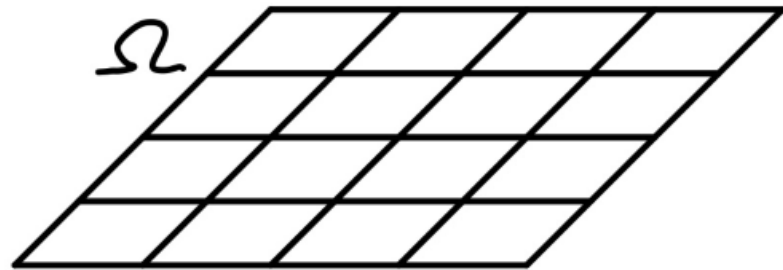
The matrix



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

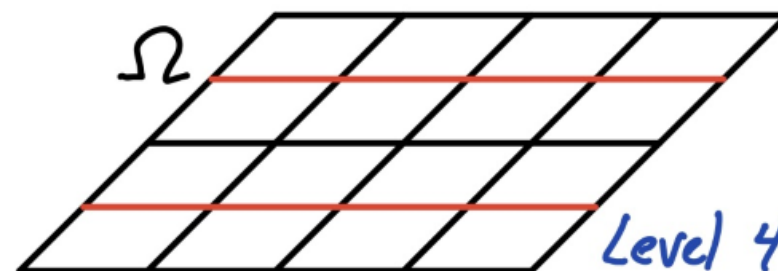
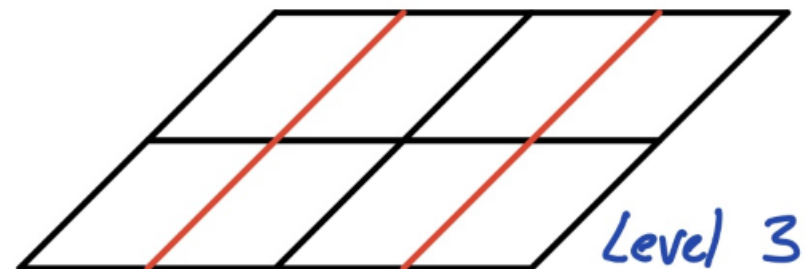
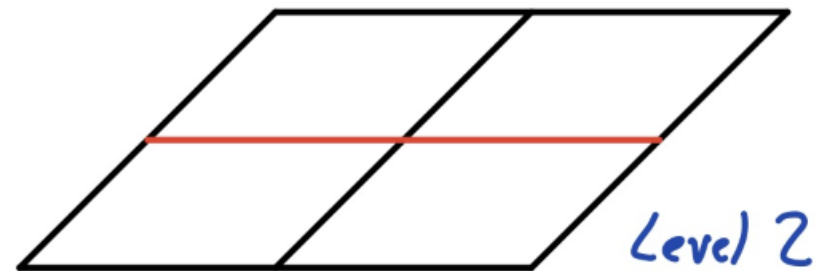
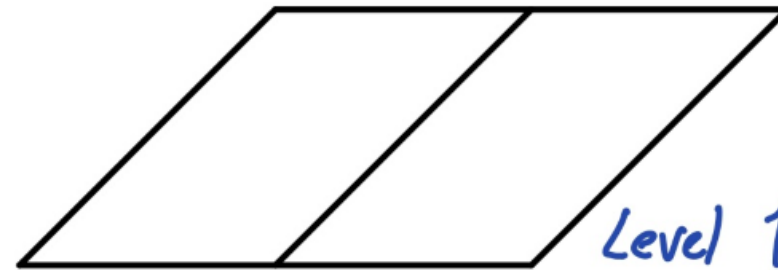
Flat tessellations

Use a single tessellation of the domain.



Hierarchical tessellations

Use a hierarchy of tessellations.

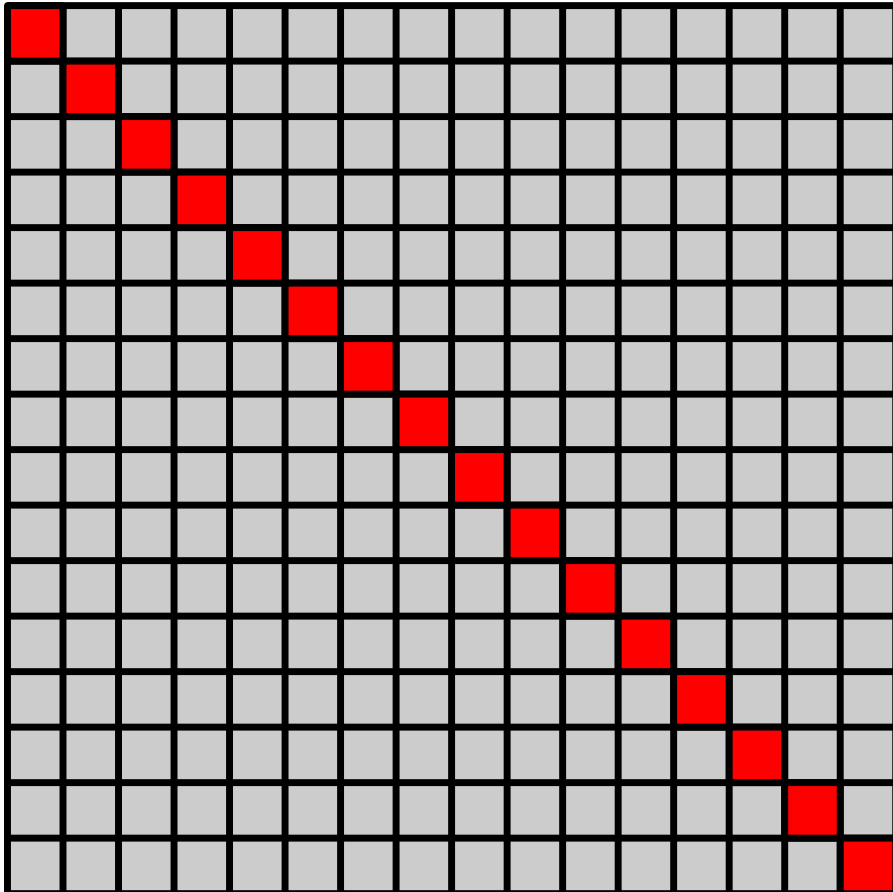


Versions of fast direct solvers: “flat” versus “hierarchical” tessellations

Flat tessellations

Use a single tessellation of the domain.

The resulting matrix:



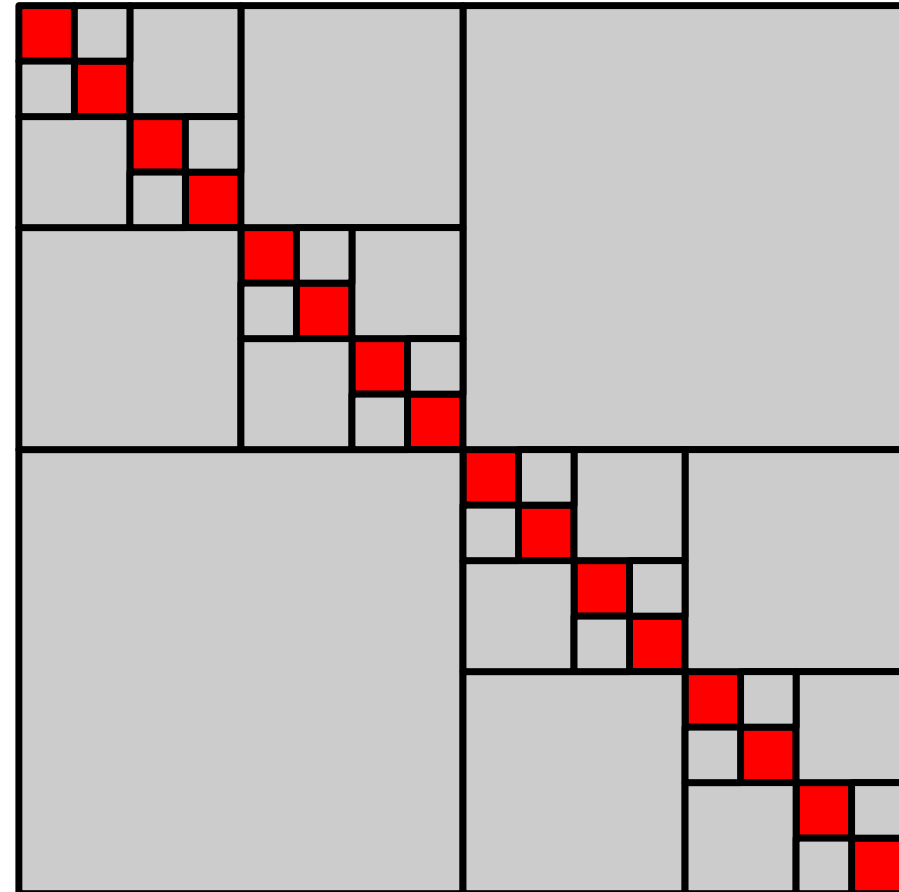
Easy to work with.

Sometimes “good enough”.

Hierarchical tessellations

Use a hierarchy of tessellations.

The resulting matrix:

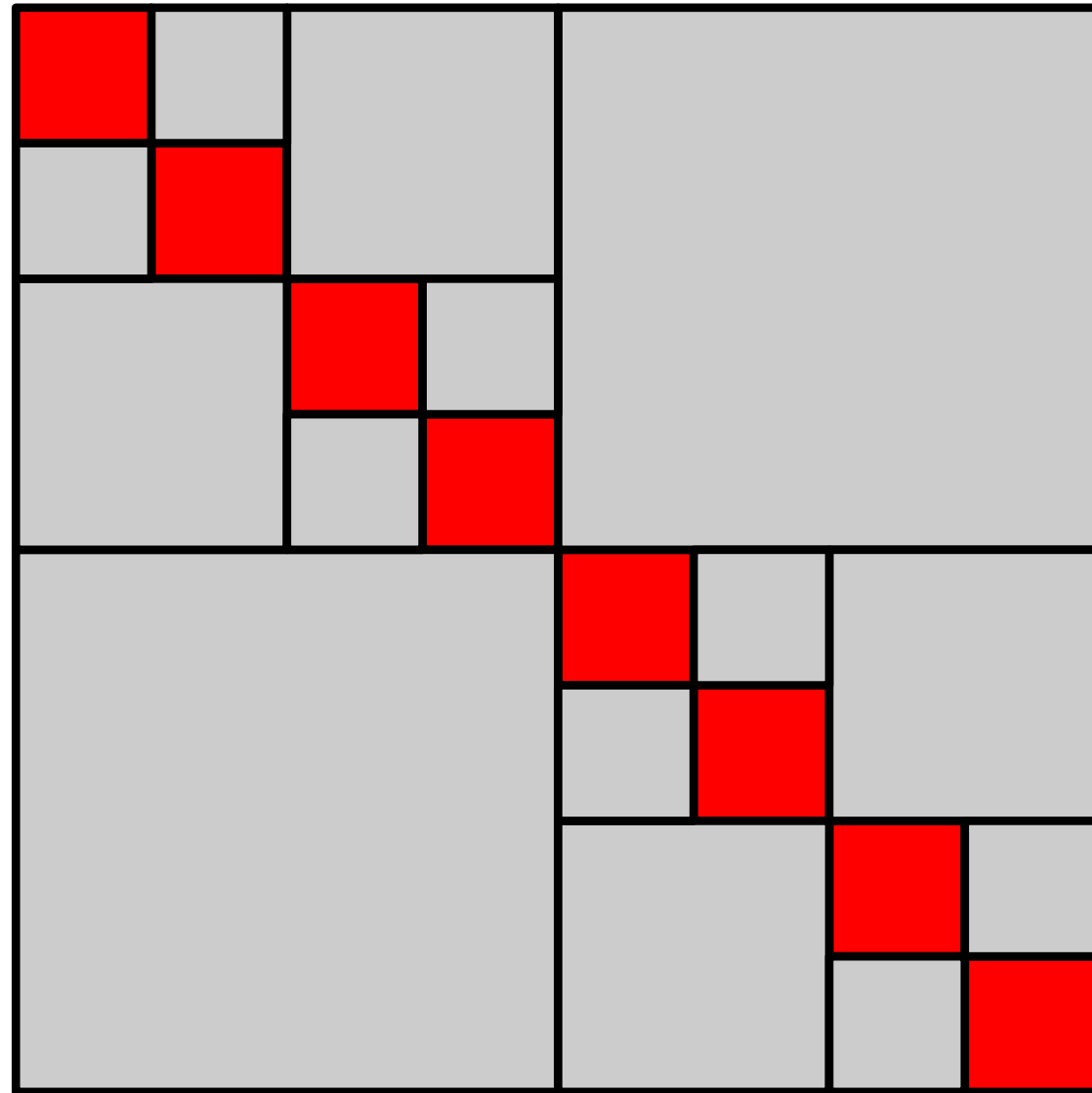


More complicated to code and analyze.

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:

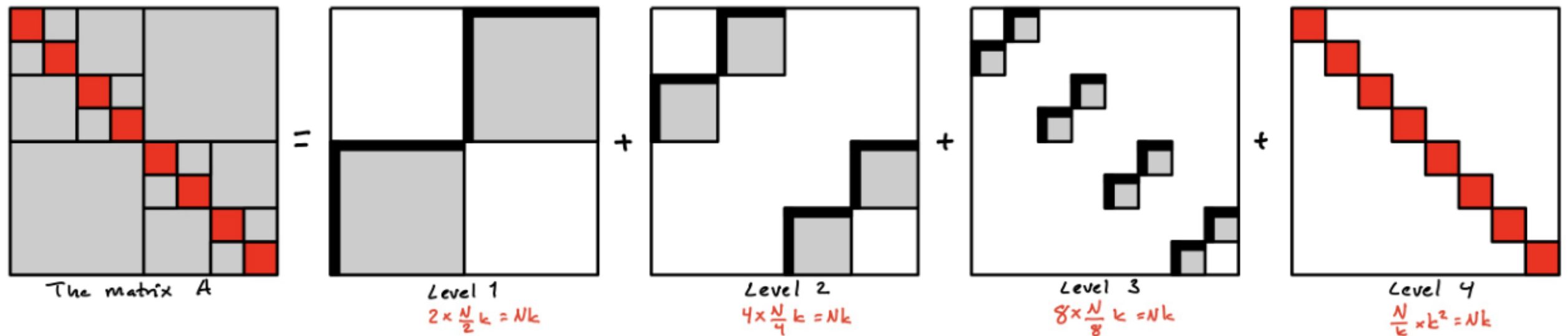


Question: How much storage is required?

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”:



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*

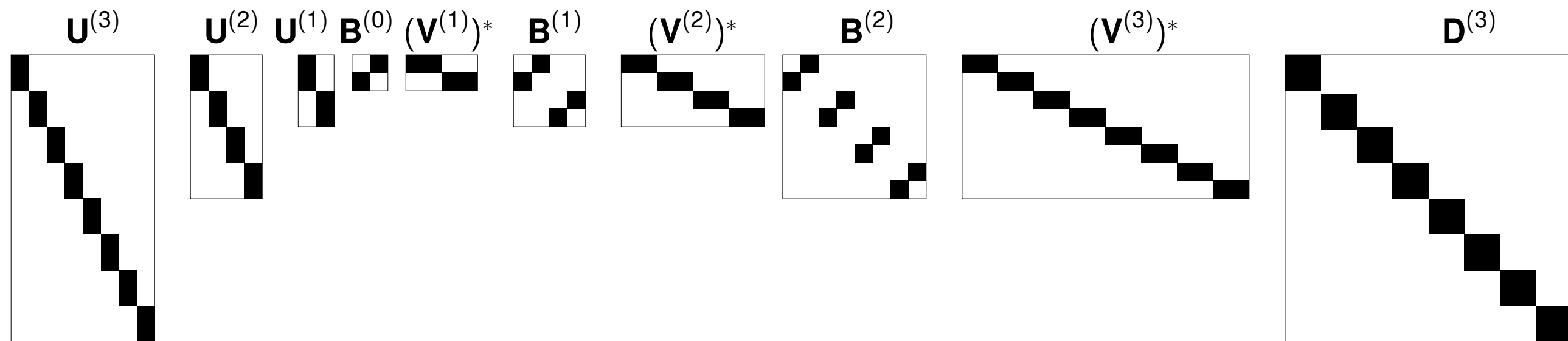
— the bases on one level are defined in terms of the bases on the next finer level.

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

For instance, it could take the form

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

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



The cost to store level ℓ is now $2^{-\ell} Nk \rightarrow$ 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.

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). *Forthcoming: Martinsson & O'Neil (2025)*

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

Inversion of matrices with “weak admissibility”

- Quick facts about weak admissibility.
- Optimal asymptotic complexity vs. practical speed.
- HSS matrices.

Quick facts about “weak admissibility”

All off-diagonal blocks are compressed into low rank form.

Ranks are *higher* than in strong admissibility.

Better inversion algorithms than what exists for strong admissibility. *Often exact.*

For problems in 1D and 2D, weak admissibility often works great.

For 3D problems, weak admissibility often works fine for surface BIEs.

In more general geometries, ranks can get quite large, and you may want to swap to strong admissibility.

Formats based on weak admissibility include: HODLR, HSS, hierarchically block separable, etc.

Inversion of 2×2 block matrix with low-rank off-diagonal blocks

Review!

Consider a 2×2 blocked matrix of size $2n \times 2n$

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

Suppose the off-diagonal blocks are rank-deficient

$$\begin{array}{ccccc} \mathbf{A}_{12} & = & \mathbf{U}_1 & \tilde{\mathbf{A}}_{12} & \mathbf{V}_2^* \\ n \times n & & n \times k & k \times k & k \times n \end{array} \quad \text{and} \quad \begin{array}{ccccc} \mathbf{A}_{21} & = & \mathbf{U}_2 & \tilde{\mathbf{A}}_{21} & \mathbf{V}_1^* \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

where $k \ll n$. We can then write \mathbf{A} as follows

$$\mathbf{A} = \underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} \end{bmatrix}}_{\text{"easy" to invert}} + \underbrace{\begin{bmatrix} \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_{12} \\ \tilde{\mathbf{A}}_{21} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^* \end{bmatrix}}_{\text{low rank}}.$$

Recall the Woodbury formula

$$(\mathbf{D} + \mathbf{U}\tilde{\mathbf{A}}\mathbf{V}^*)^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{U}(\tilde{\mathbf{A}} + \mathbf{V}^*\mathbf{D}^{-1}\mathbf{U})^{-1}\mathbf{V}^*\mathbf{D}^{-1}.$$

Applying the Woodbury formula, we find, with $\hat{\mathbf{D}}_{11} = \mathbf{V}_1^*\mathbf{A}_{11}^{-1}\mathbf{U}_1$ and $\hat{\mathbf{D}}_2 = \mathbf{V}_2^*\mathbf{A}_{22}^{-1}\mathbf{U}_2$,

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

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 $n \times n$:

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

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

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

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).

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

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

$$\mathbf{A} = \mathbf{U} \tilde{\mathbf{A}} \mathbf{V}^* + \mathbf{D},$$

$pn \times pn$ $pn \times pk$ $pk \times pk$ $pk \times pn$ $pn \times pn$

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

$$\mathbf{A} = \mathbf{U} \tilde{\mathbf{A}} \mathbf{V}^* + \mathbf{D},$$

$pn \times pn$ $pn \times pk$ $pk \times pk$ $pk \times pn$ $pn \times pn$

then

$$\mathbf{A}^{-1} = \mathbf{E} (\tilde{\mathbf{A}} + \hat{\mathbf{D}})^{-1} \mathbf{F}^* + \mathbf{G},$$

$pn \times pn$ $pn \times pk$ $pk \times pk$ $pk \times pn$ $pn \times pn$

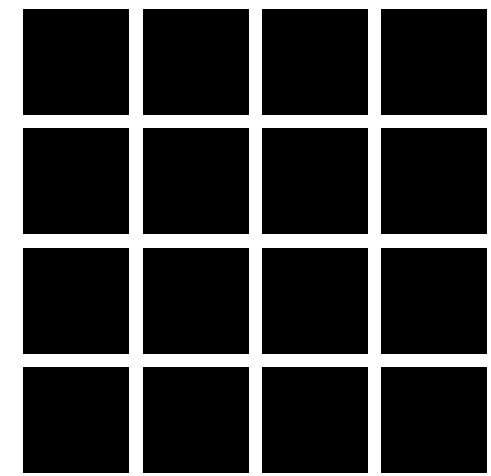
where (provided all intermediate matrices are invertible)

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

Note: All matrices set in blue are block diagonal.

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

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



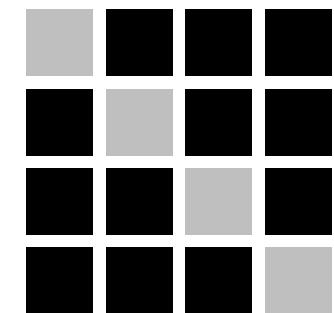
The original $4n \times 4n$ matrix.

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 $\hat{\mathbf{D}}_{ii} = (\mathbf{V}_i^* \mathbf{A}_{ii}^{-1} \mathbf{U}_i)^{-1}$.
- 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 $\tilde{\mathbf{A}}_{i,j} = \mathbf{A}(\tilde{l}_i, \tilde{l}_j)$.

The Interpolative Decomposition (ID):

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

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

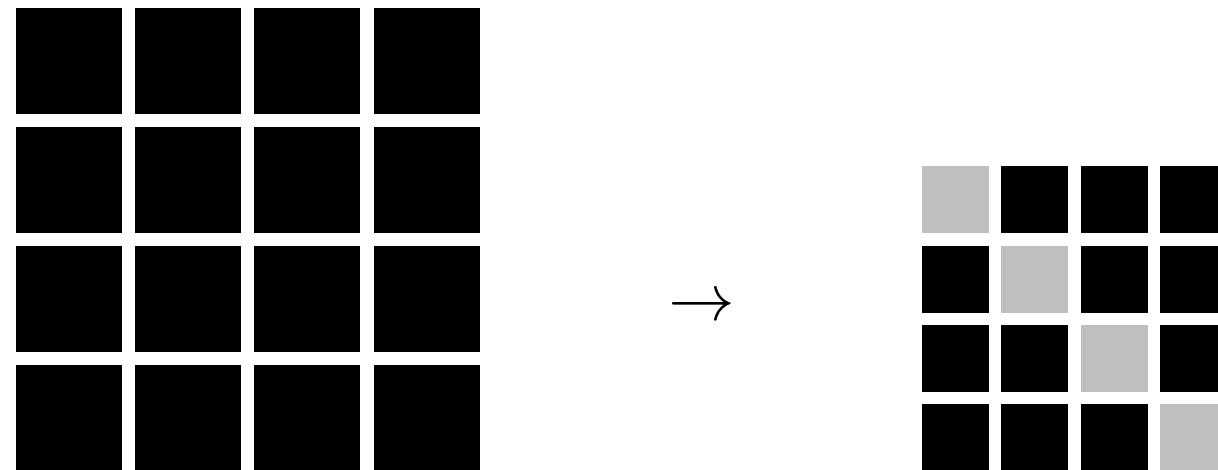
where

1. $\mathbf{B}^{(\text{skel})} = \mathbf{B}(I_{\text{row}}, I_{\text{col}})$ 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 $pk \times pk$.



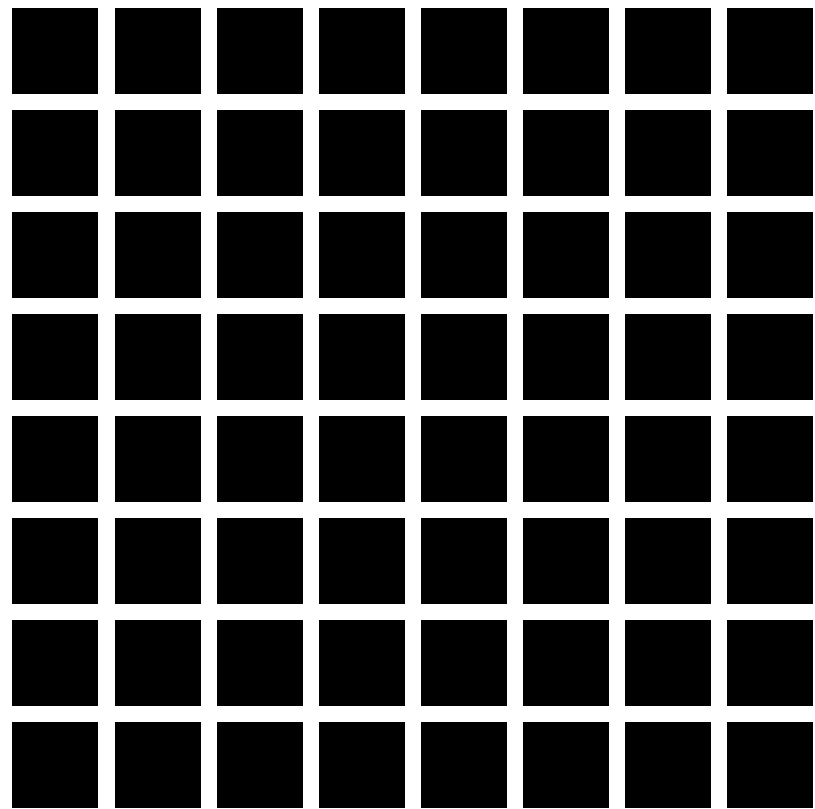
Important: The black blocks are **submatrices**. No need to compute them!

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

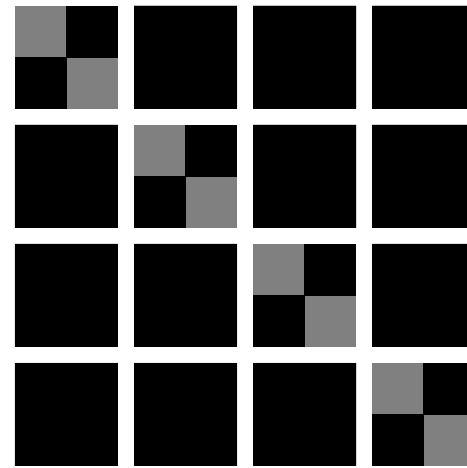
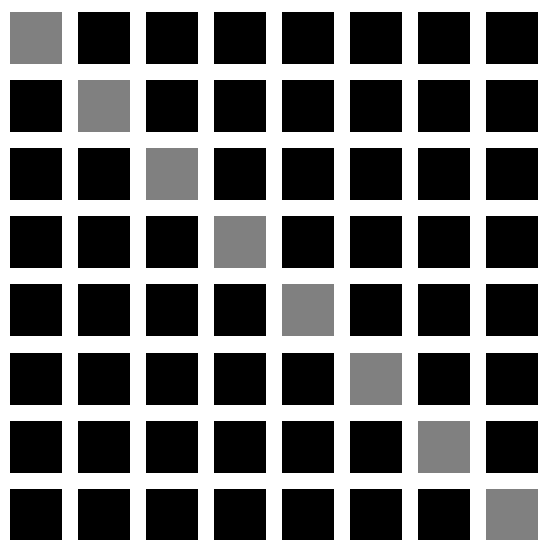
Question: How do we get to $O(N)$?

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

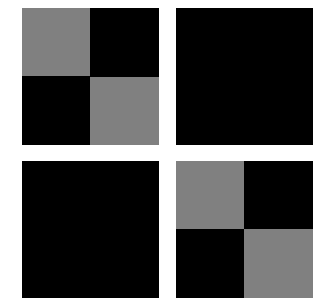
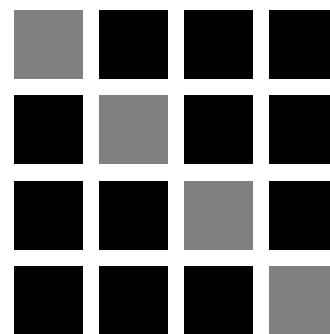
A globally $O(N)$ algorithm is obtained by hierarchically repeating the process:



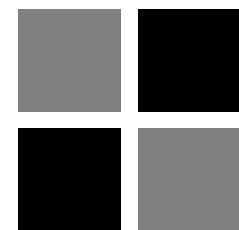
↓ Compress



↓ Compress



↓ Compress



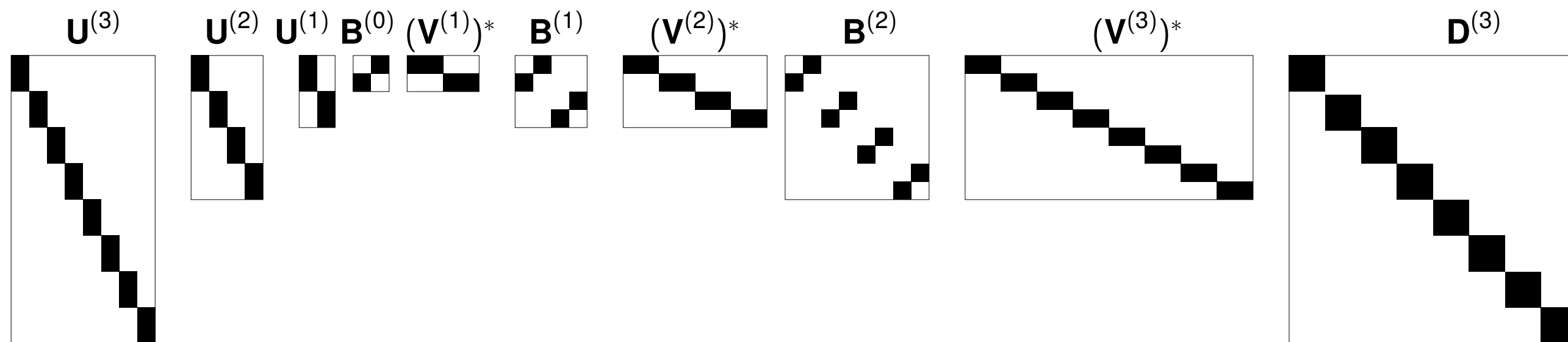
↗
Cluster

↗
Cluster

Formally, one can view this as a telescoping factorization of \mathbf{A} :

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

Expressed pictorially, the factorization takes the form



The *inverse of A* then takes the form

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

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.

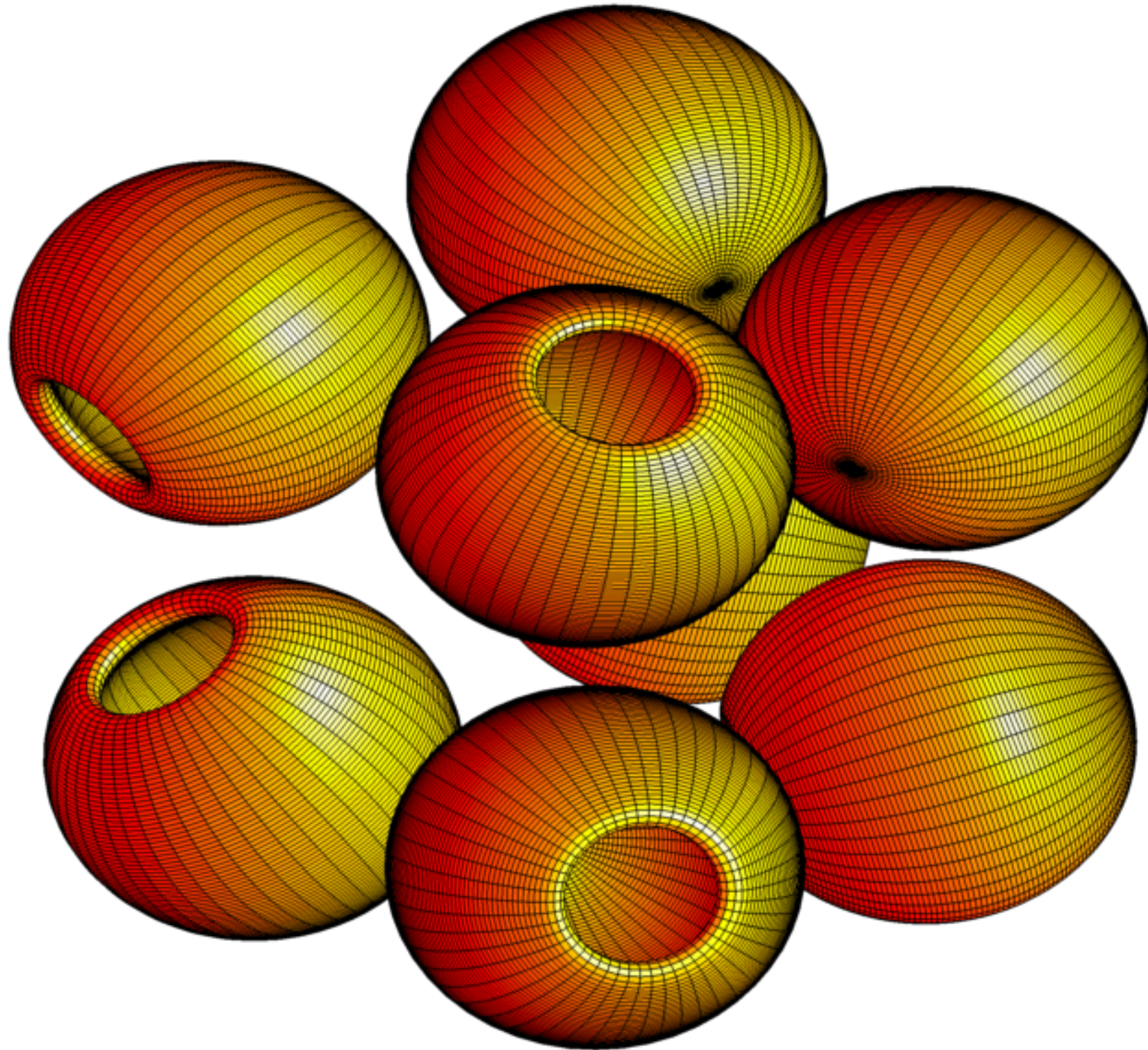
Recall:

$$\hat{\mathbf{D}}_{\mathcal{T}} = (\mathbf{V}^* \mathbf{A}_{\mathcal{T}}^{-1} \mathbf{U}_{\mathcal{T}})^{-1}$$

You can instead assemble just the “scattering matrix”

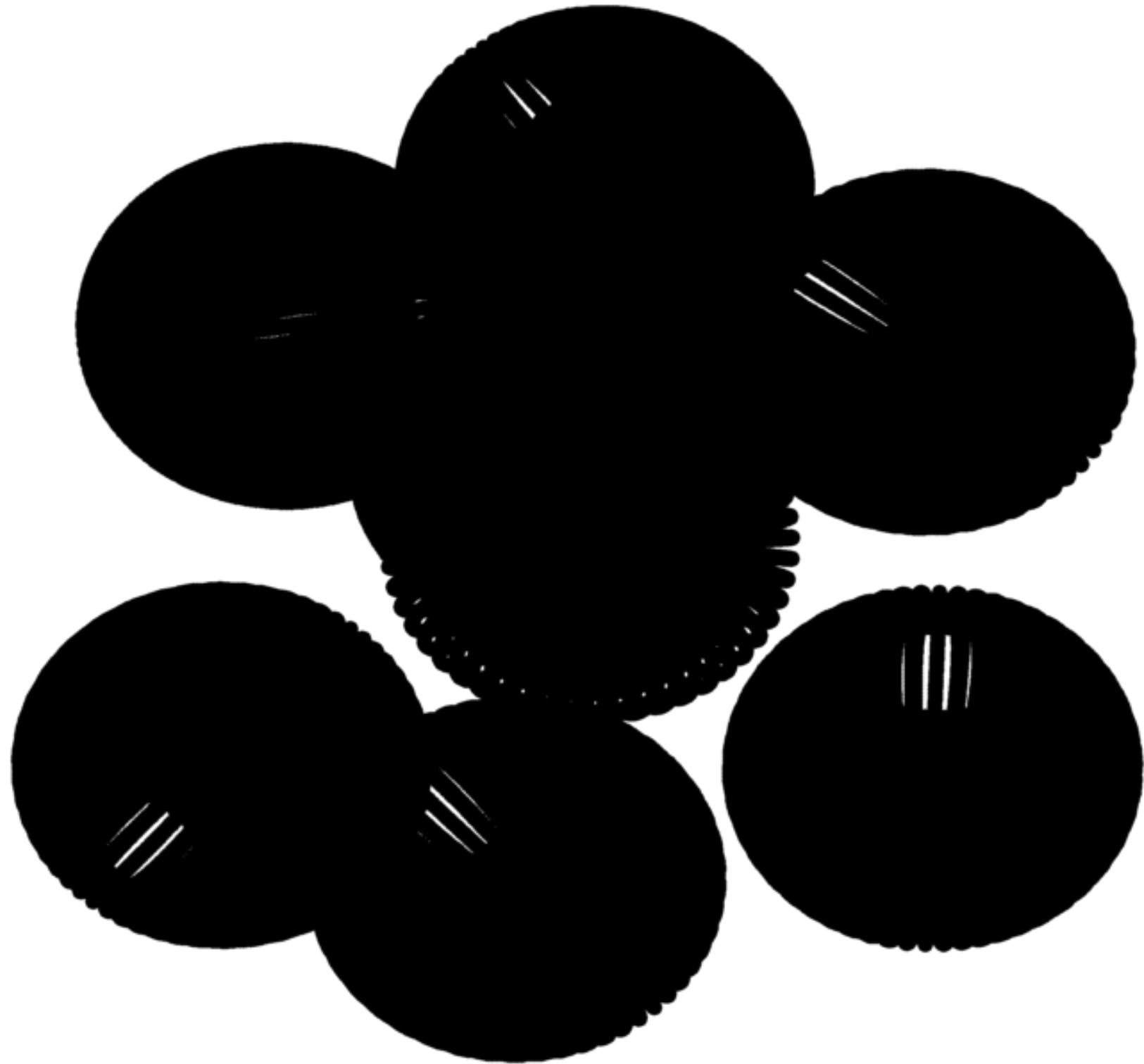
$$\mathbf{S}_{\mathcal{T}} = \mathbf{V}^* \mathbf{A}_{\mathcal{T}}^{-1} \mathbf{U}_{\mathcal{T}}.$$

Example: Multibody scattering



Consider scattering from some multibody domain involving cavities.

Example: Multibody scattering



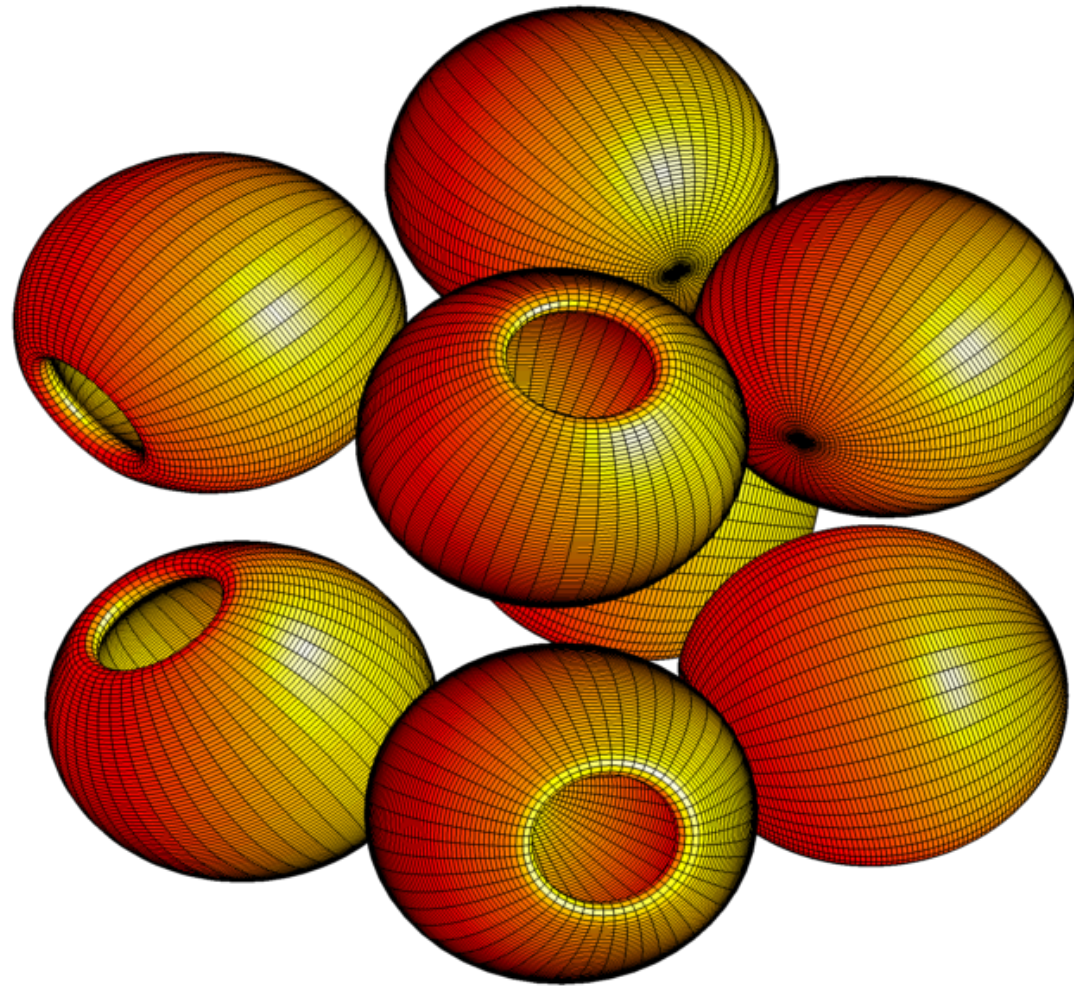
There are lots of discretization nodes involved. Very computationally intense!

Example: Multibody scattering



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

Example: Multibody scattering



Acoustic scattering on the exterior domain.

Each bowl is about 5λ .

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

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

Example: Multibody scattering

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

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

Example: Multibody scattering

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

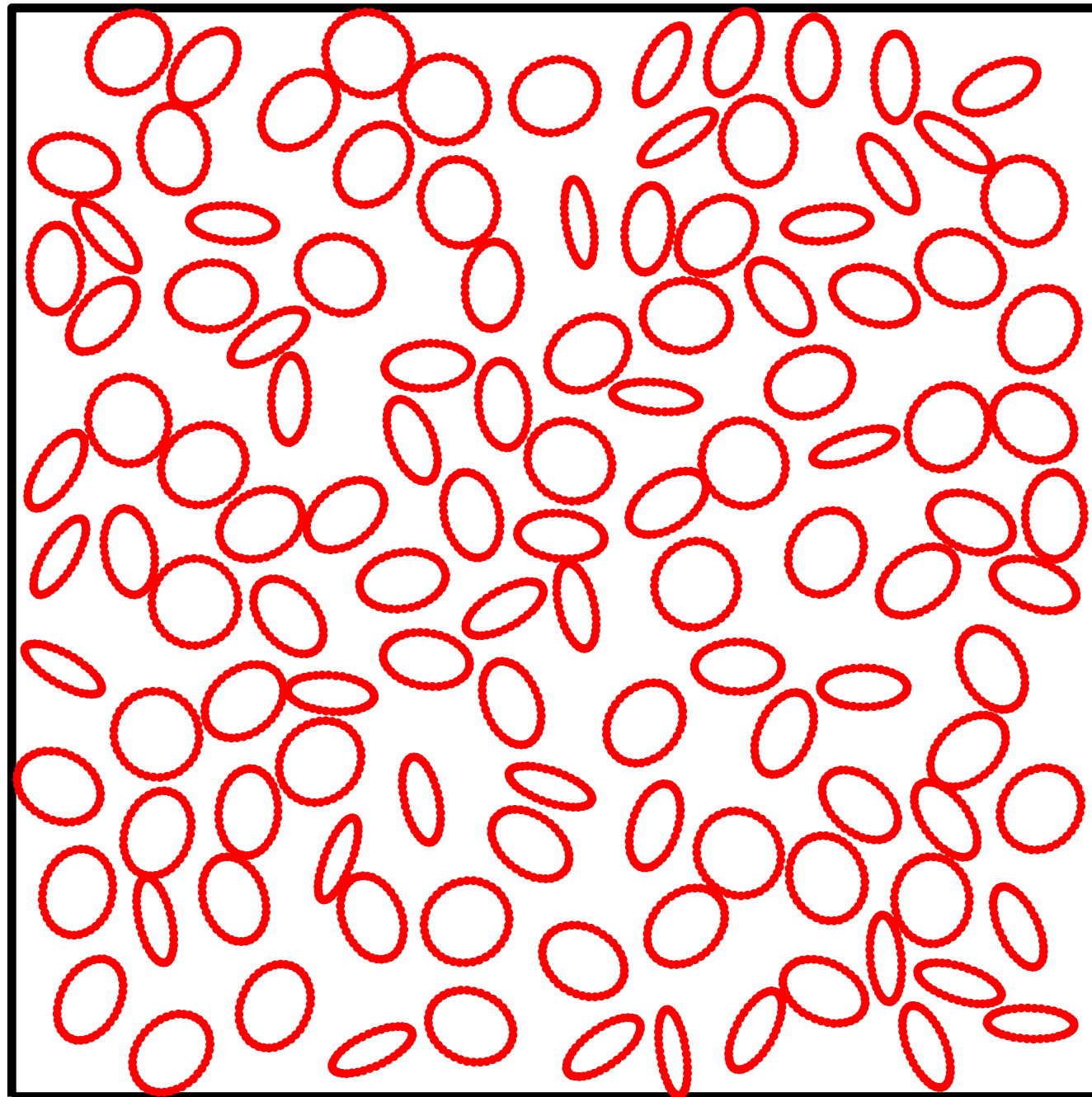
*Exterior **Helmholtz** problem solved on multibody bowl domain.*

Each bowl is 5 wavelength in diameter.

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

Example: A hierarchical direct solver

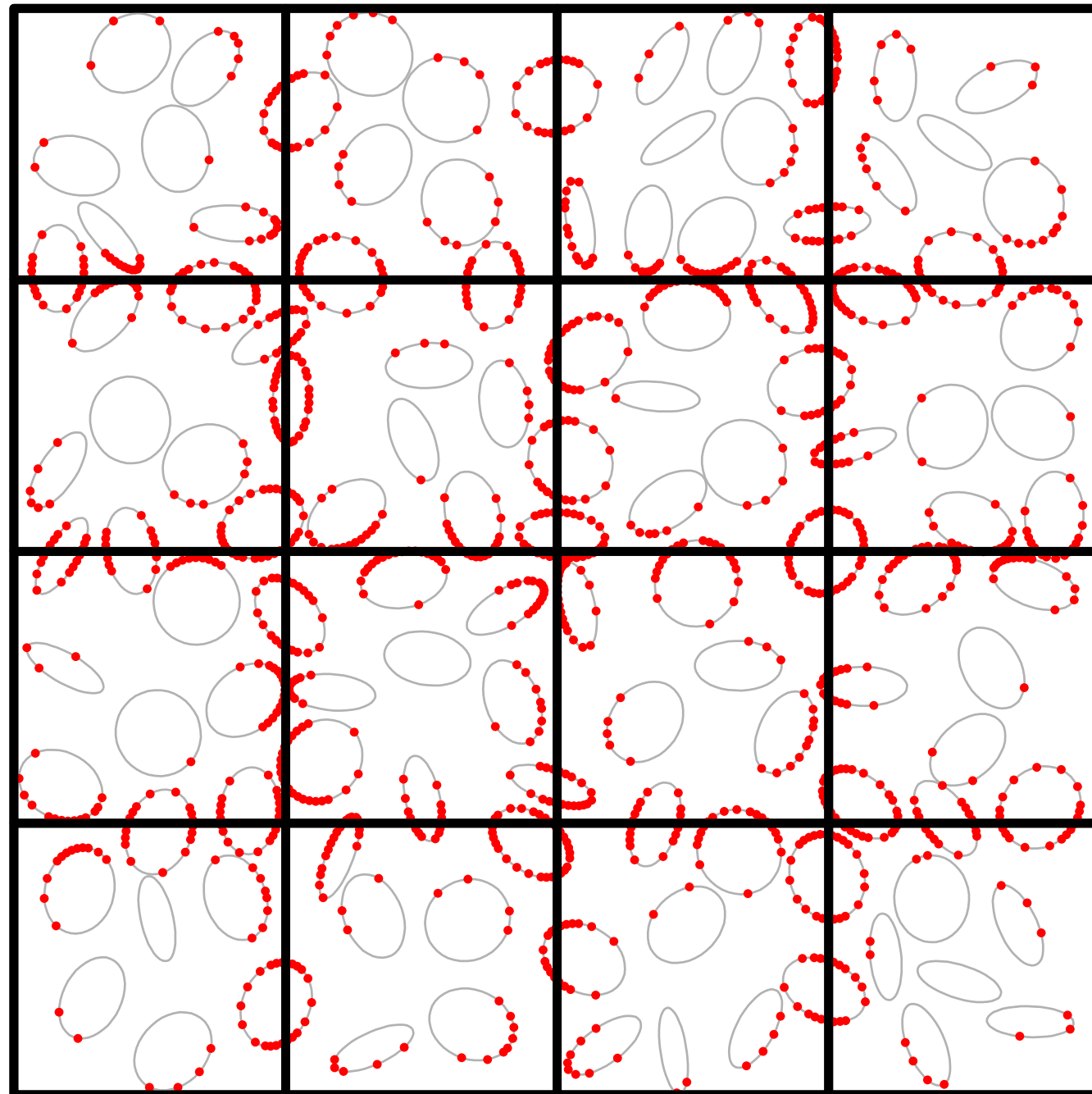
Original set of points



Consider a boundary integral equation that is defined on the collection of red contours shown in the picture above. Think of a computational model of a composite material, or the simulation of colloidal fluid.

Example: A hierarchical direct solver

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

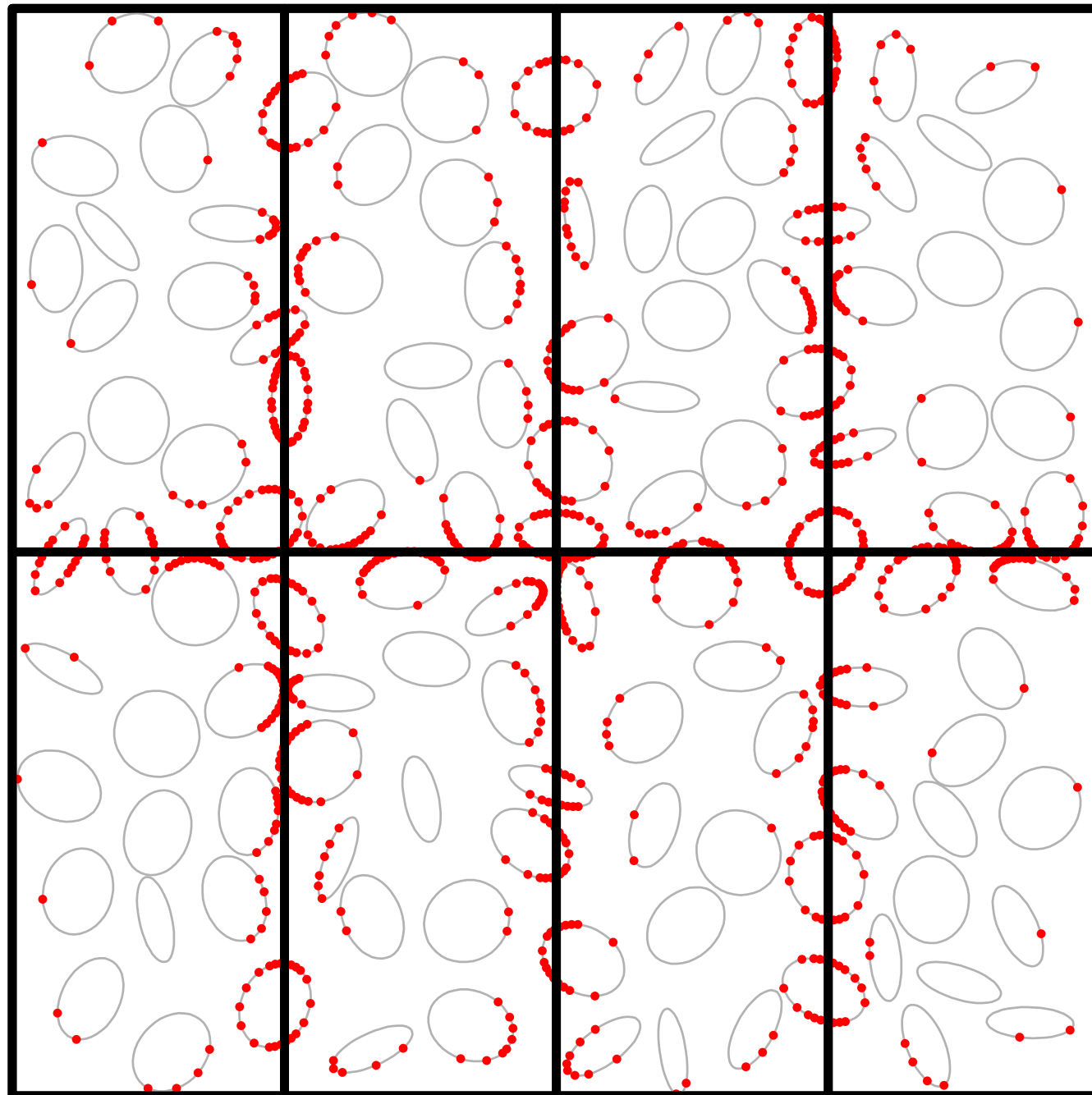


Standard expansions (multipole, Bessel functions, etc do not work!

Numerical compression is necessary.

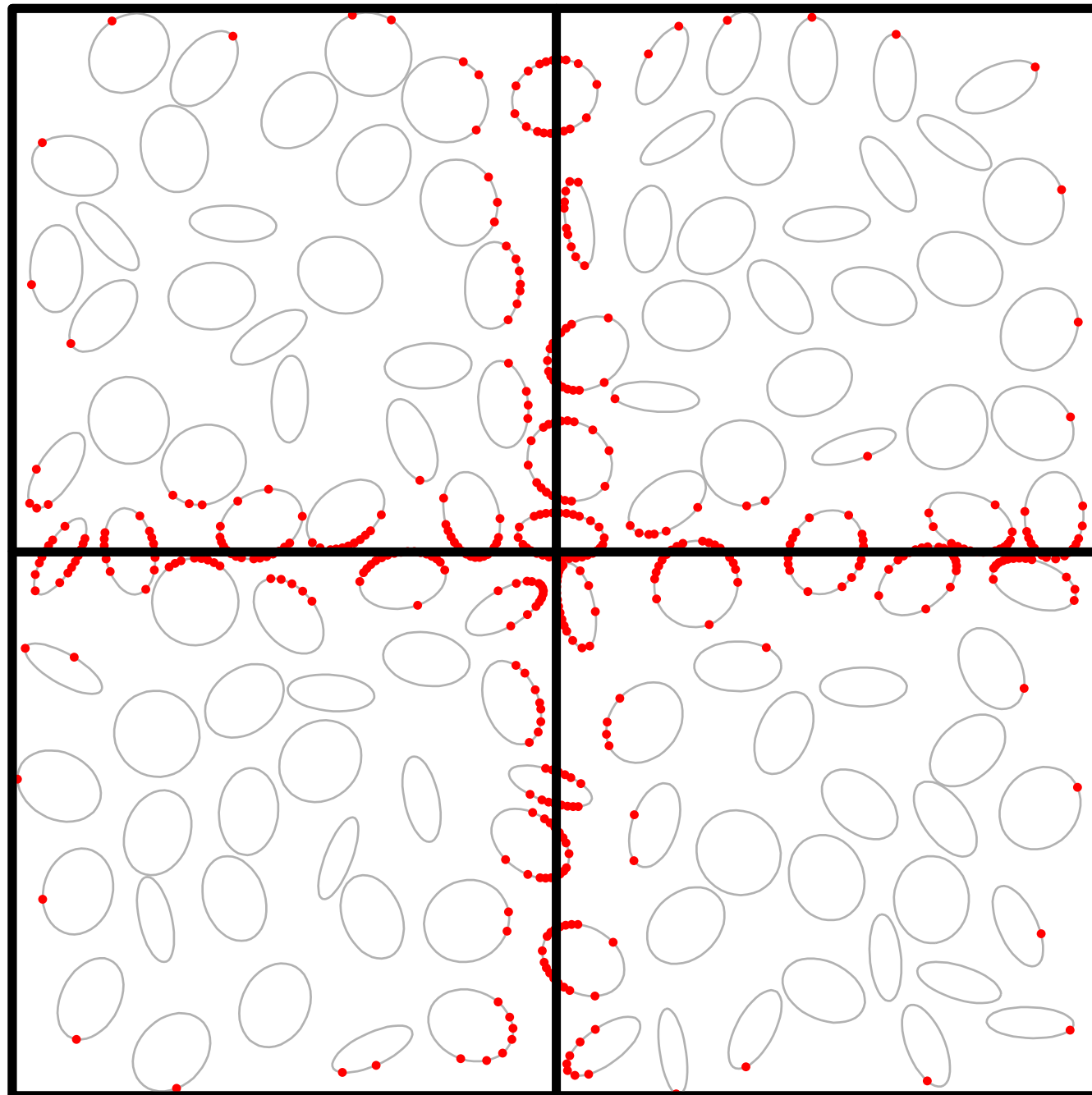
Example: A hierarchical direct solver

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



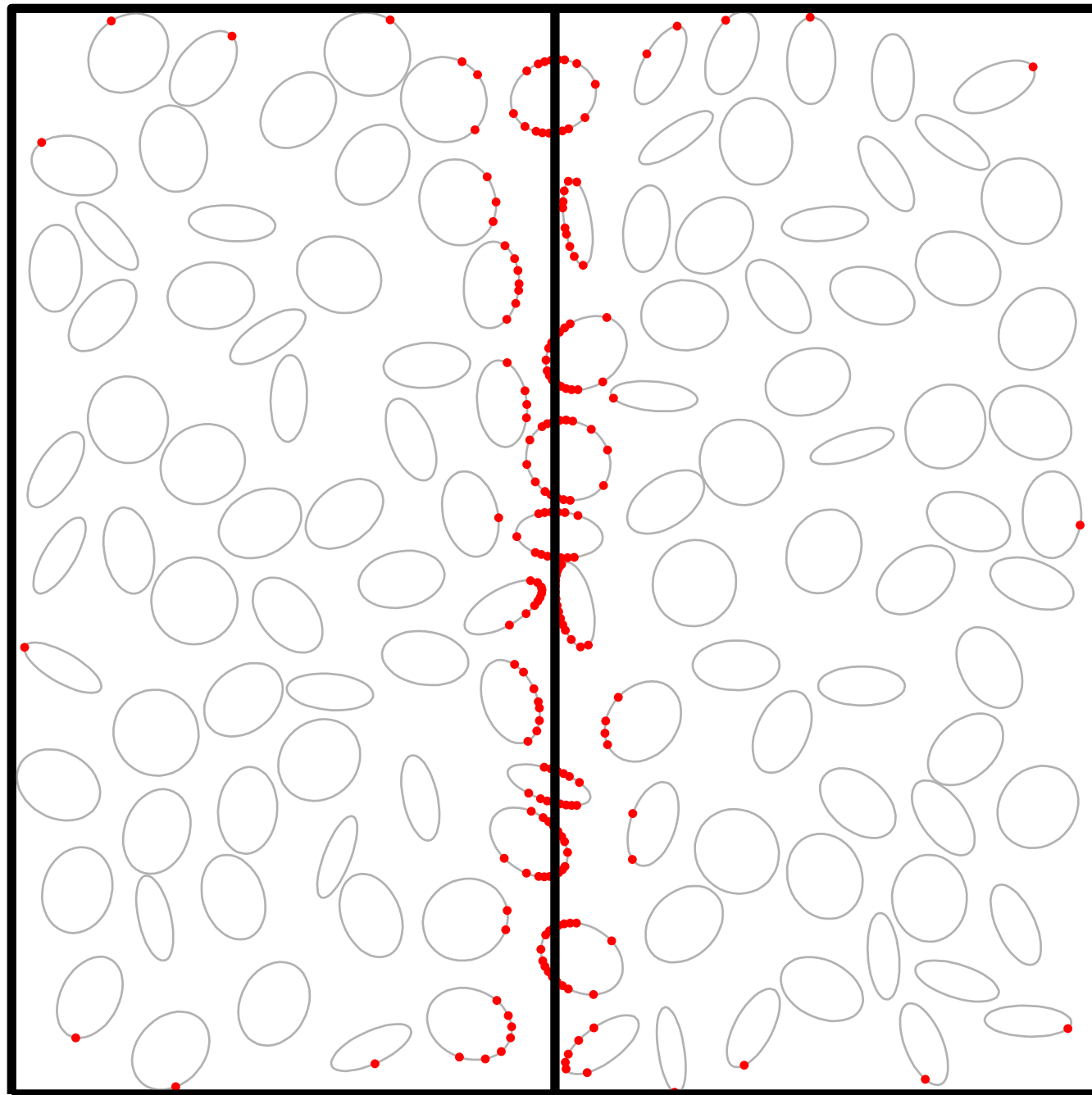
Example: A hierarchical direct solver

Skeleton points on level 2, acc = 1.000e-09

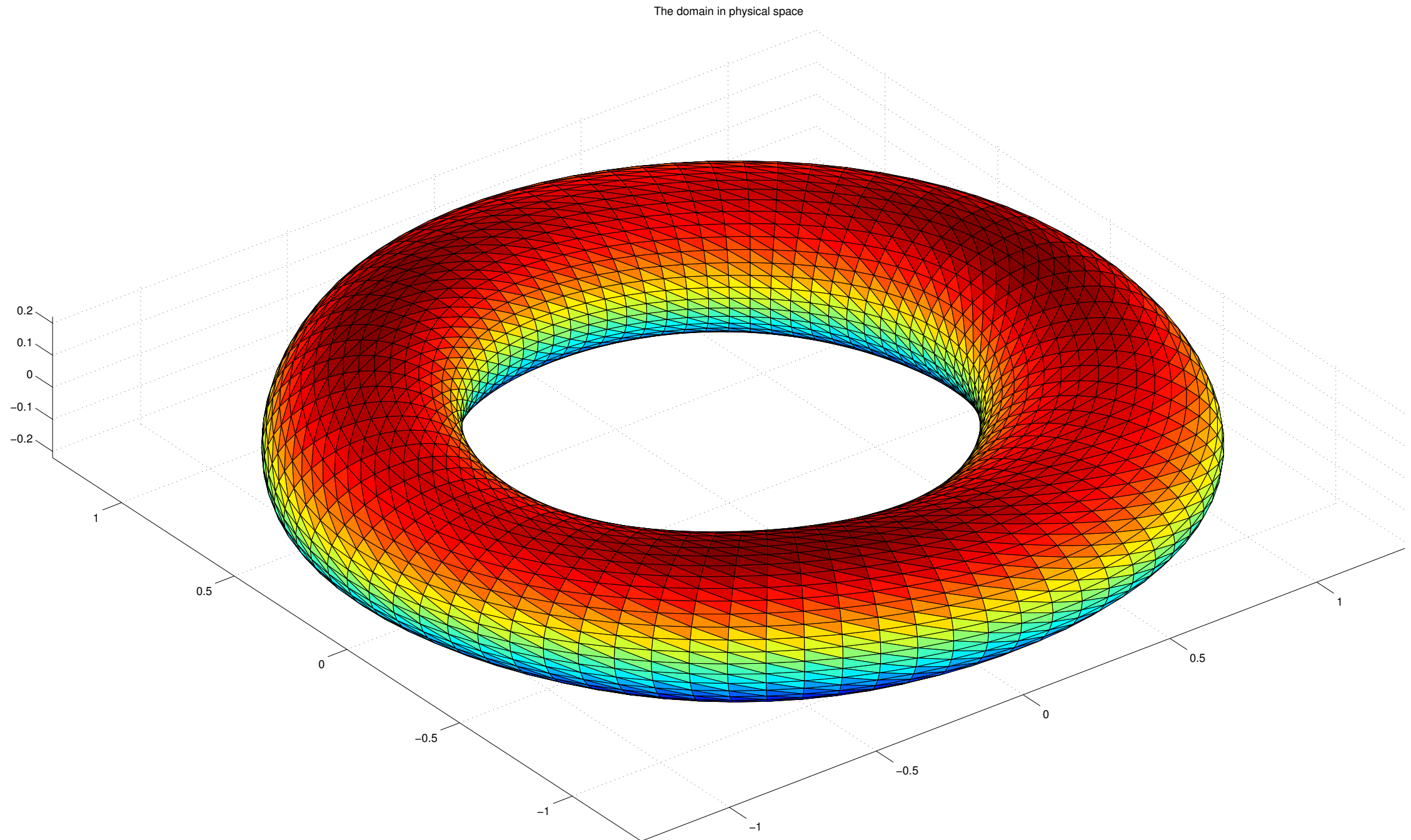


Example: A hierarchical direct solver

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



Example: Surface BIE



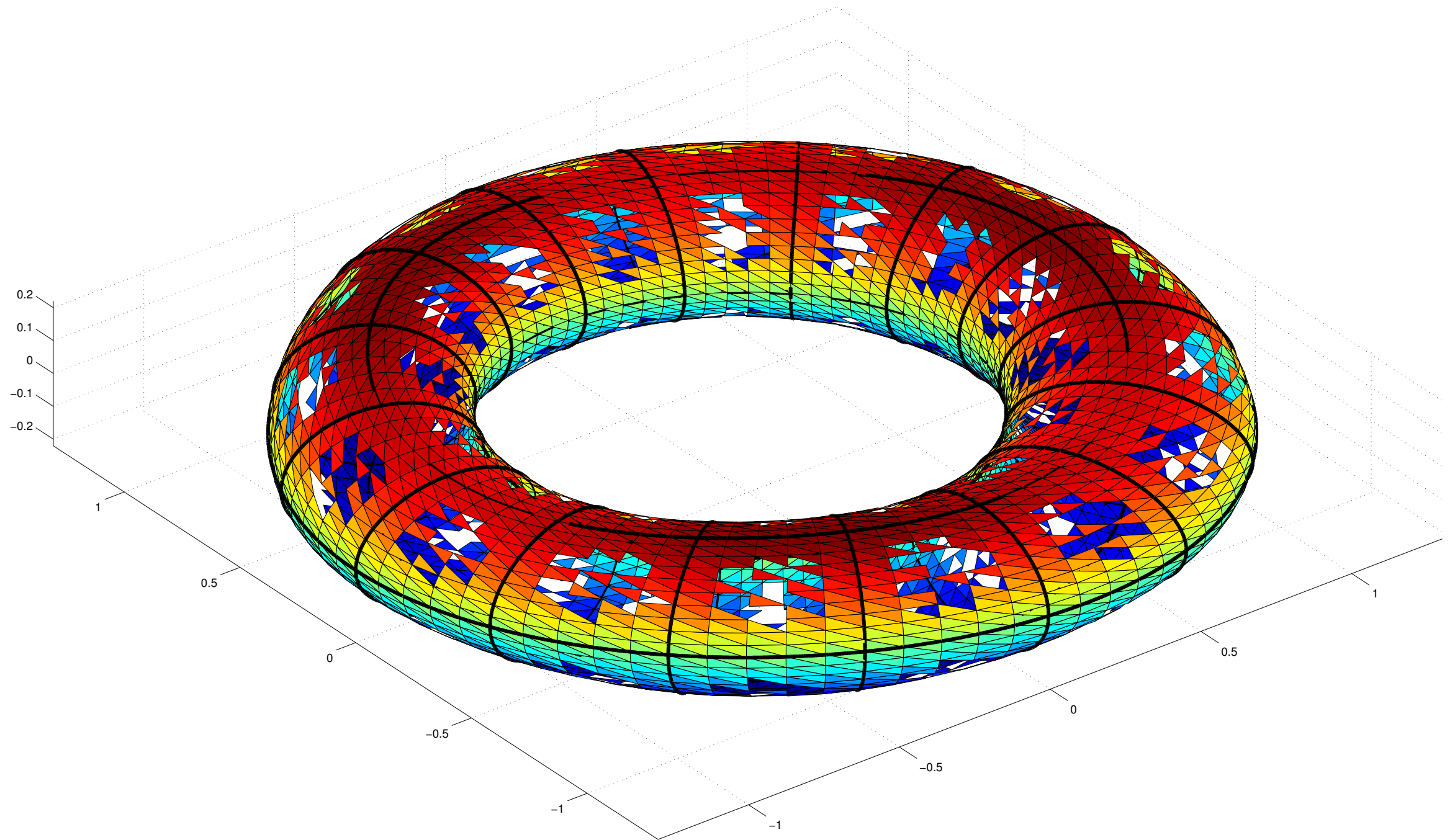
Let \mathbf{A} denote an $N \times N$ matrix arising upon discretizing a boundary integral operator

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

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

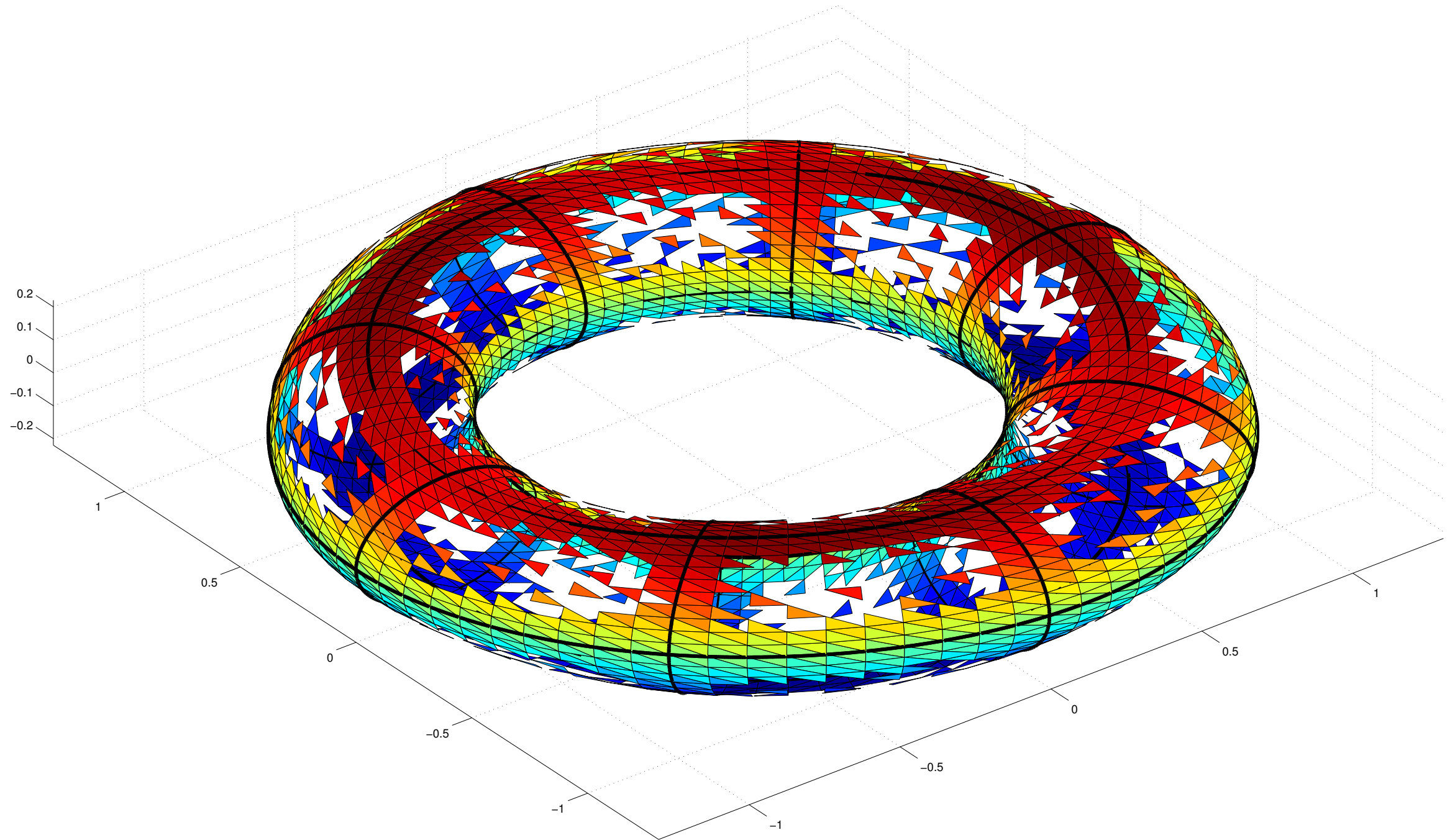
Example: Surface BIE

The domain in physical space



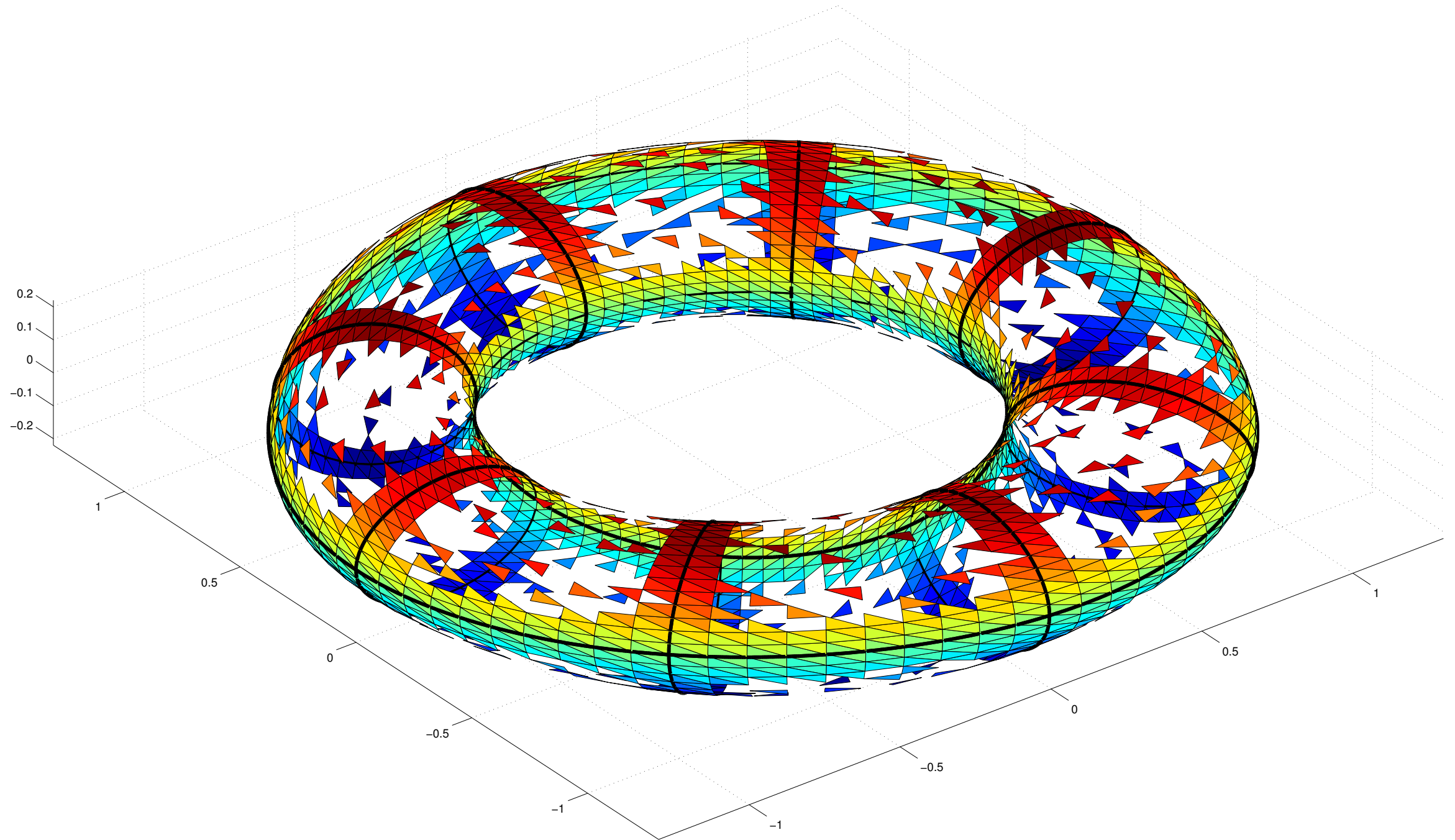
Example: Surface BIE

The domain in physical space



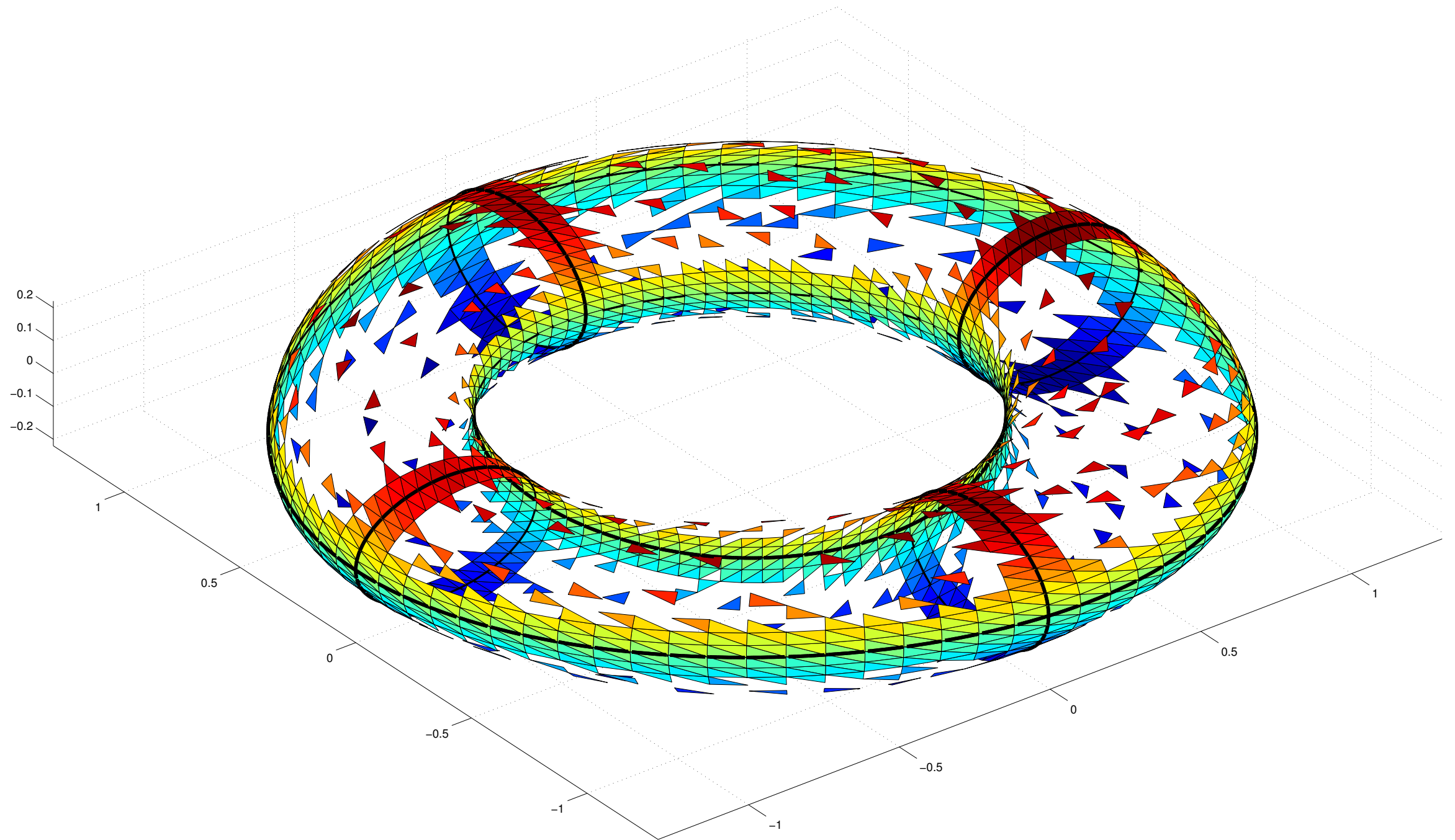
Example: Surface BIE

The domain in physical space



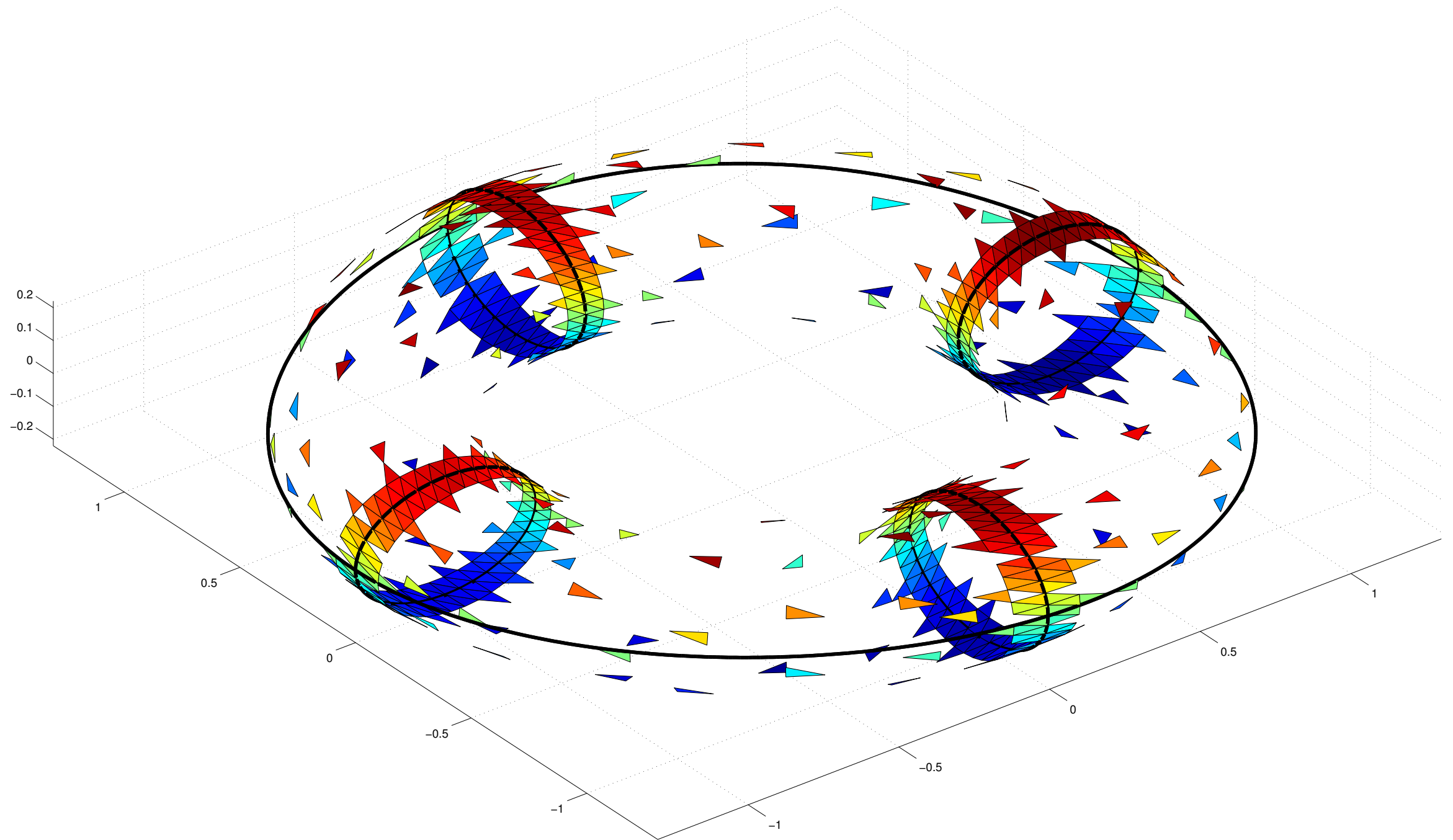
Example: Surface BIE

The domain in physical space



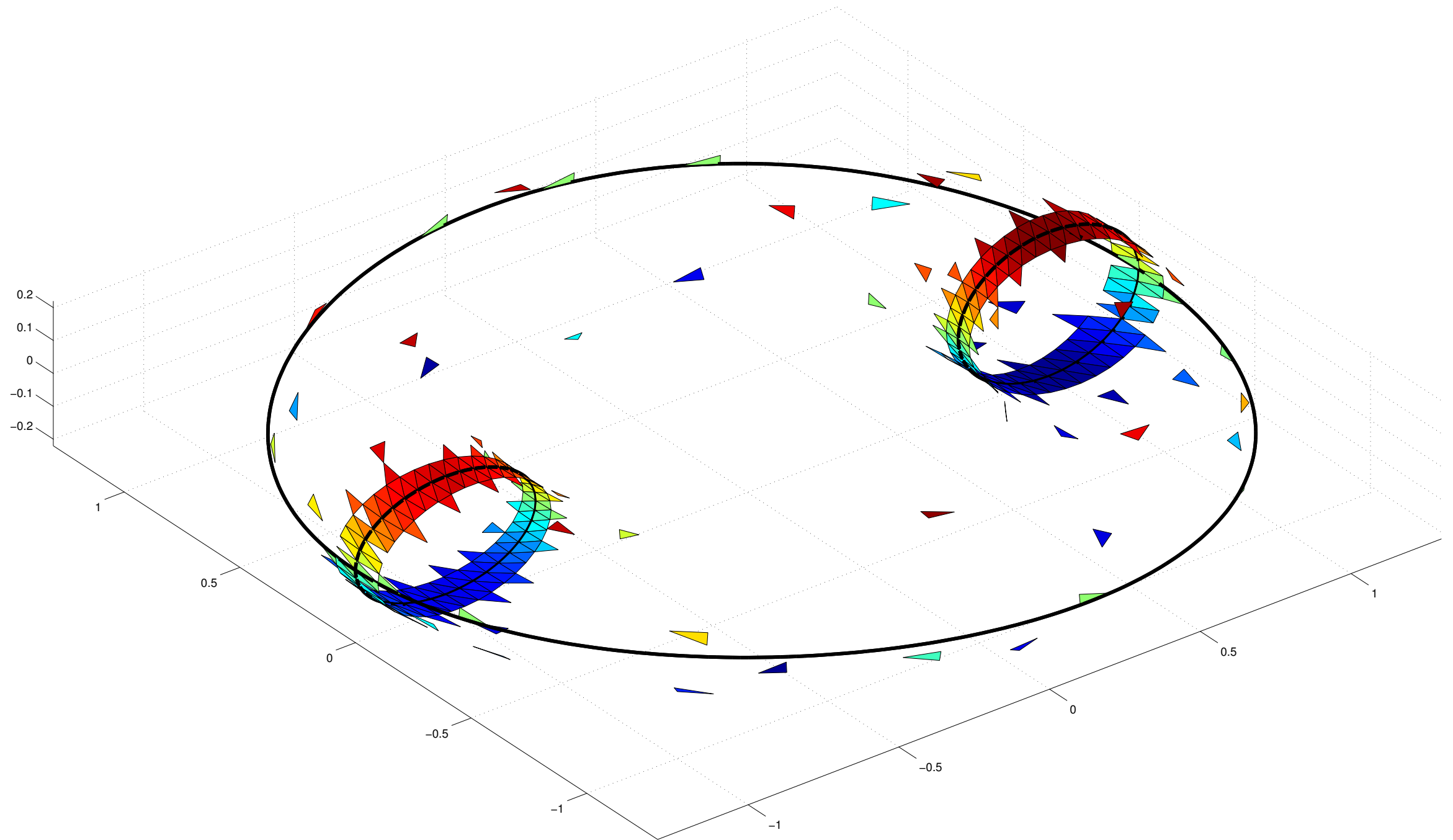
Example: Surface BIE

The domain in physical space



Example: Surface BIE

The domain in physical space



Constructing a data-sparse representation of an integral operator

The talk has described a number of different ways that rank structured matrices can be inverted or factorized.

But how do you obtain the low rank representation in the first place?

Constructing a data-sparse representation of an integral operator

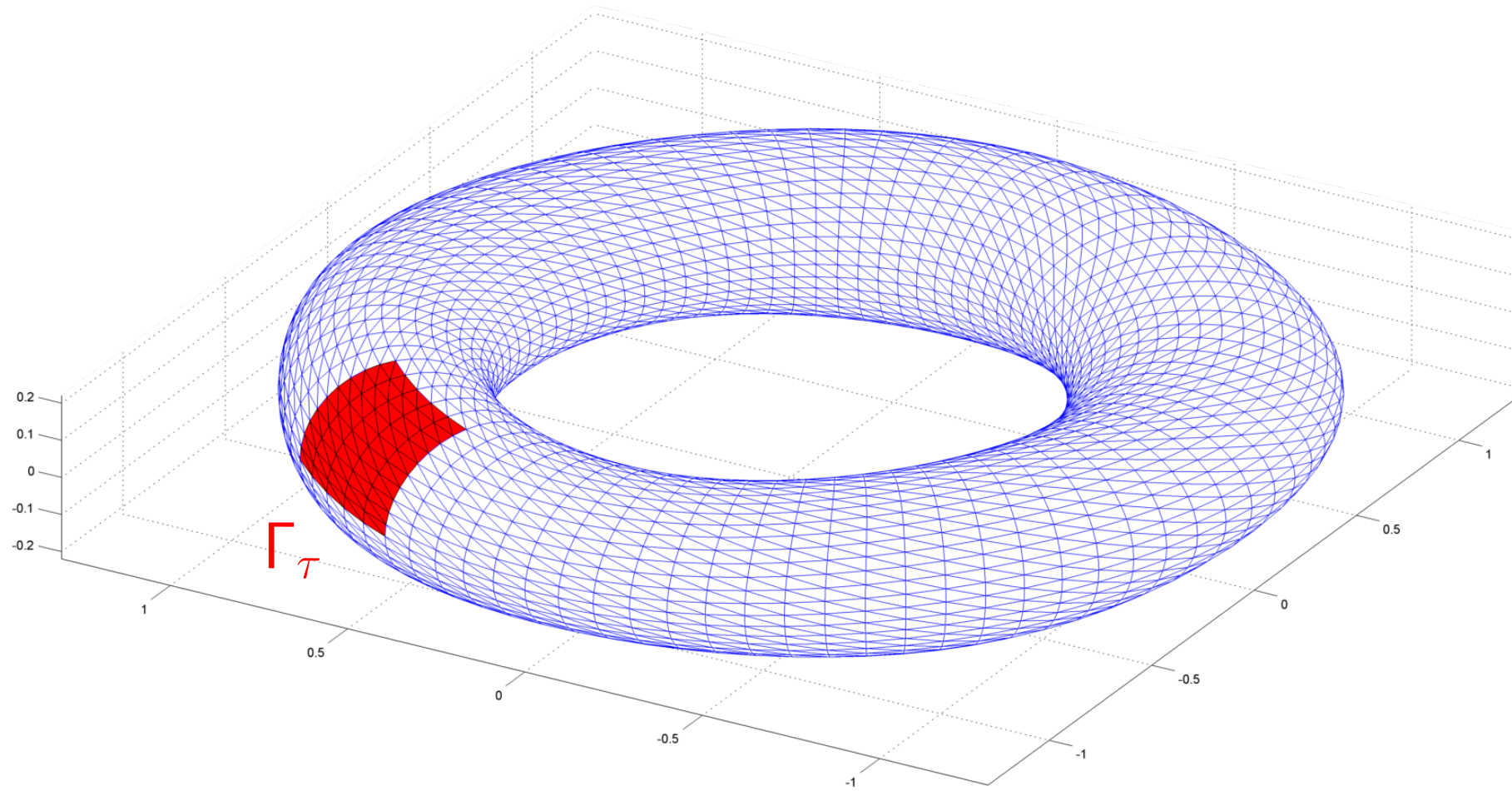
The talk has described a number of different ways that rank structured matrices can be inverted or factorized.

But how do you obtain the low rank representation in the first place?

Several methods have been proposed over the years:

- Taylor / Chebyshev expansions. *Does not work for weak admissibility.*
- Multipole expansions. *Does not work for weak admissibility.*
- Adaptive Cross Approximation. Perform completely pivoted Gaussian elimination on a cleverly selected subset of rows and columns of the matrix. Very efficient. Can yield $O(1)$ failure without it being detected. Ok for pre-conditioning.
- Randomized compression. There exist randomized algorithms that can construct a data sparse representation of a rank structured matrix \mathbf{A} by observing only its action on vectors $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$. Works very well for weak admissibility! *[Wed!]*
- Proxy surfaces. Replace all far field interactions with interactions through a set of “proxy points”. Exploits potential theory.

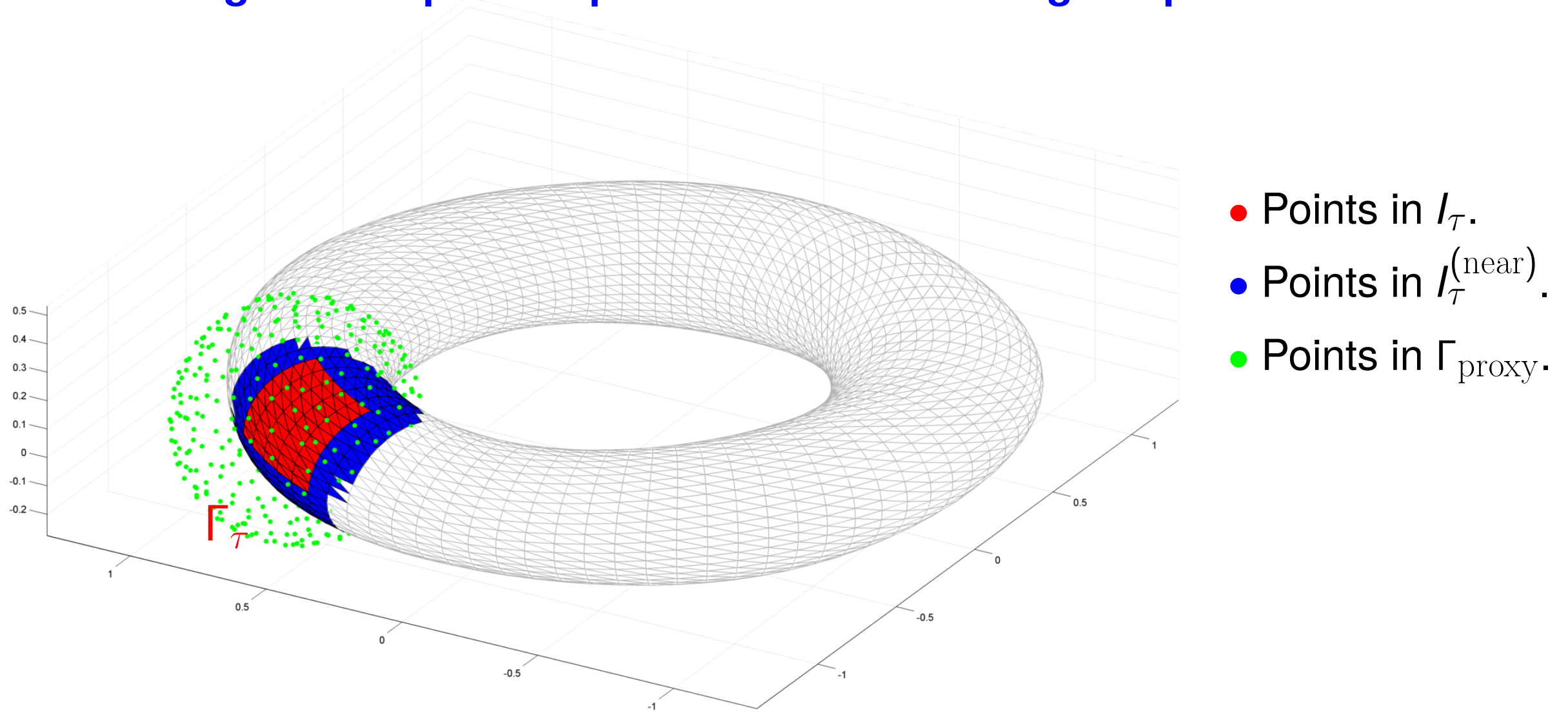
Constructing a data-sparse representation of an integral operator



- Points in $I_{\mathcal{T}}$.
- Points in $I_{\mathcal{T}}^c$.

At first, it seems like we need to perform an ID of the large matrix $\mathbf{A}(I_{\mathcal{T}}, I_{\mathcal{T}}^c)$.

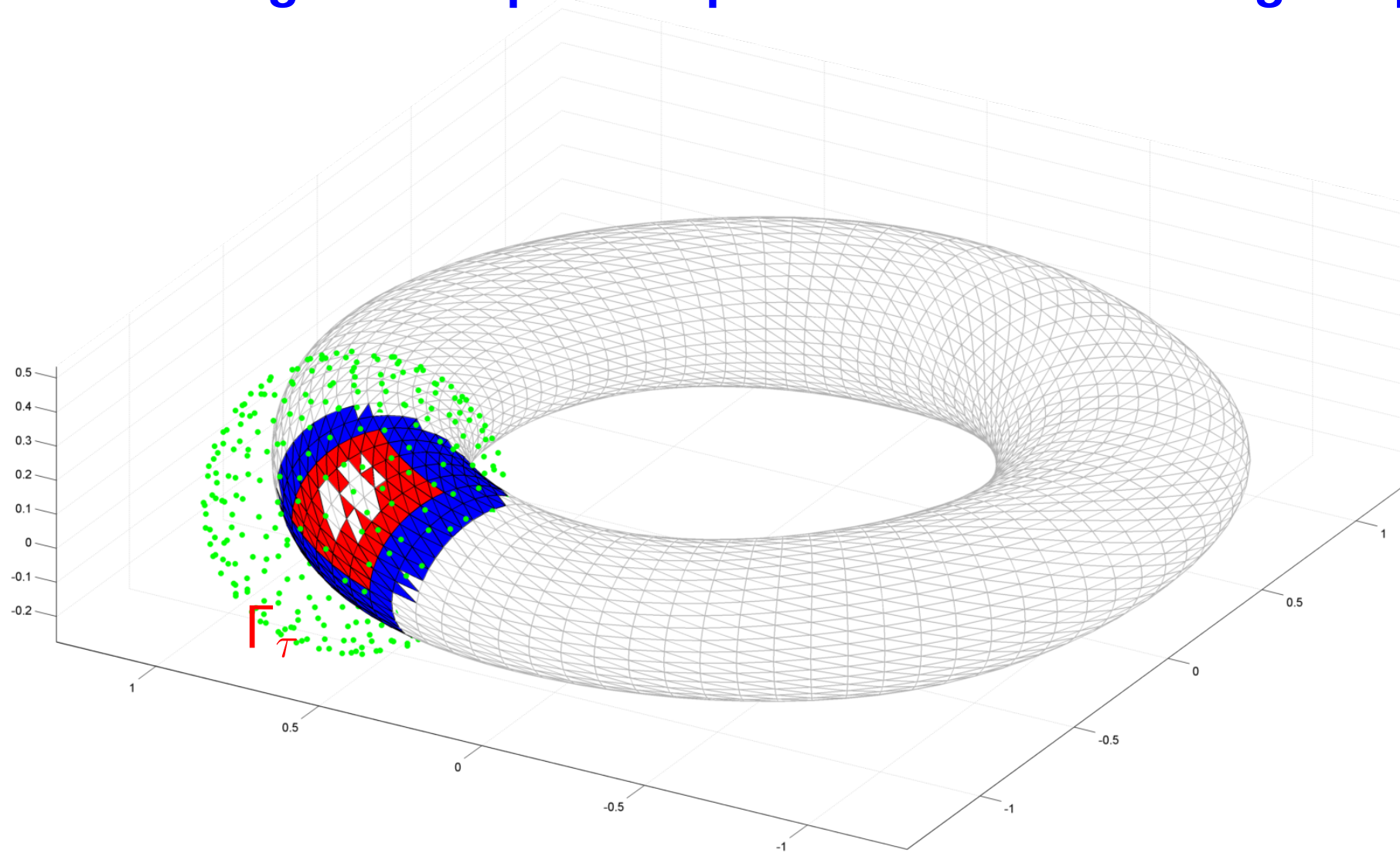
Constructing a data-sparse representation of an integral operator



At first, it seems like we need to perform an ID of the large matrix $\mathbf{A}(I_T, I_T^c)$.

But, using the *Green localization trick*, we only need to ID the matrix $[\mathbf{A}(I_T, I_T^{(\text{near})}) \mathbf{G}]$, where \mathbf{G} is the matrix of interaction with the proxy surface (green).

Constructing a data-sparse representation of an integral operator



- Points in $\tilde{I}_{\mathcal{T}}$.
- Points in $I_{\mathcal{T}}^{(\text{near})}$.
- Points in Γ_{proxy} .

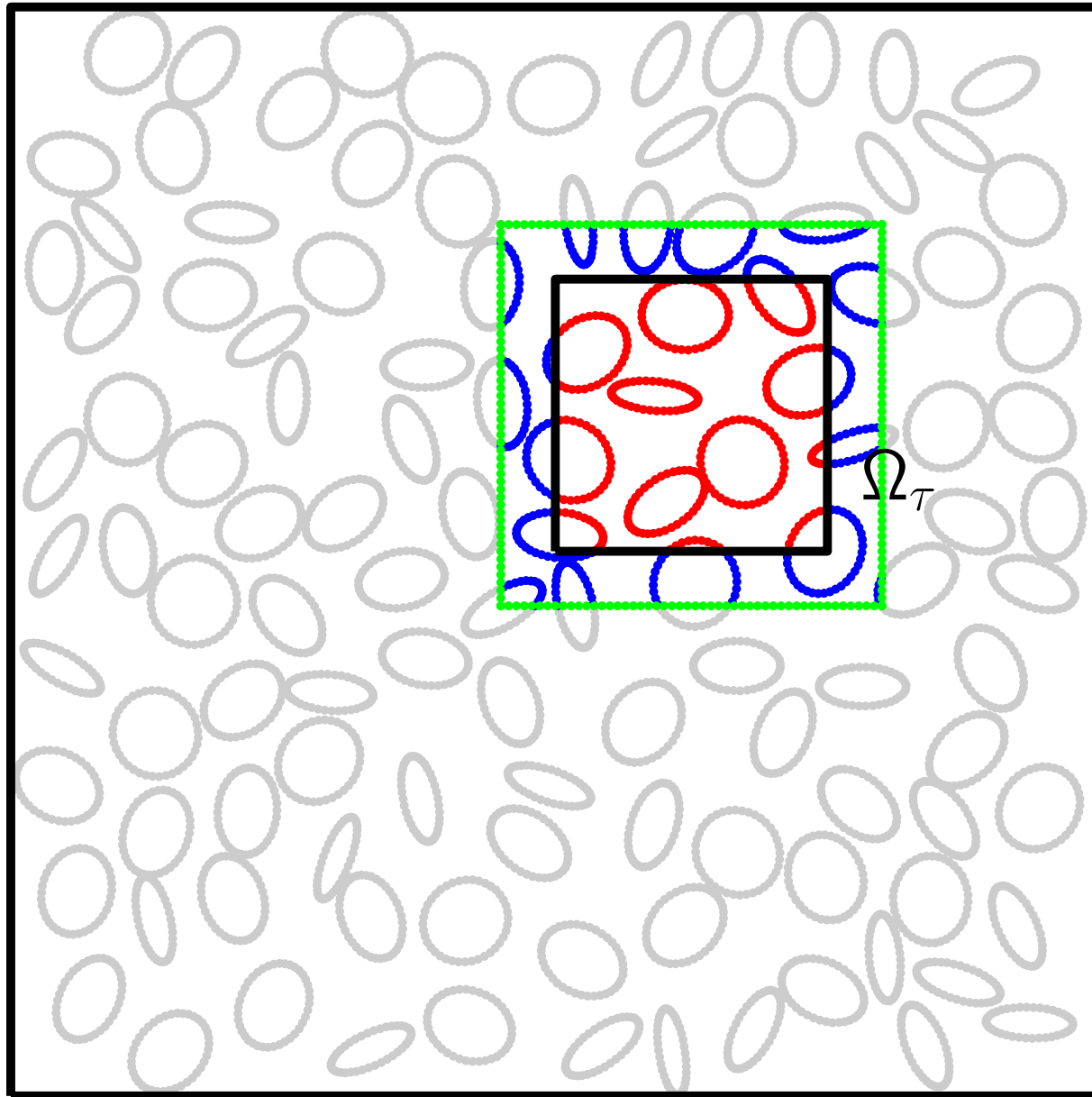
At first, it seems like we need to perform an ID of the large matrix $\mathbf{A}(I_{\mathcal{T}}, I_{\mathcal{T}}^c)$.

But, using the *Green localization trick*, we only need to ID the matrix $[\mathbf{A}(I_{\mathcal{T}}, I_{\mathcal{T}}^{(\text{near})}) \mathbf{G}]$, where \mathbf{G} is the matrix of interaction with the proxy surface (green).

Constructing a data-sparse representation of an integral operator

Illustration of the proxy point compression technique:

When compressing a subdomain I_τ (shown in red), we replace all “far-field” points (gray) by a set of artificial “proxy points” (green).

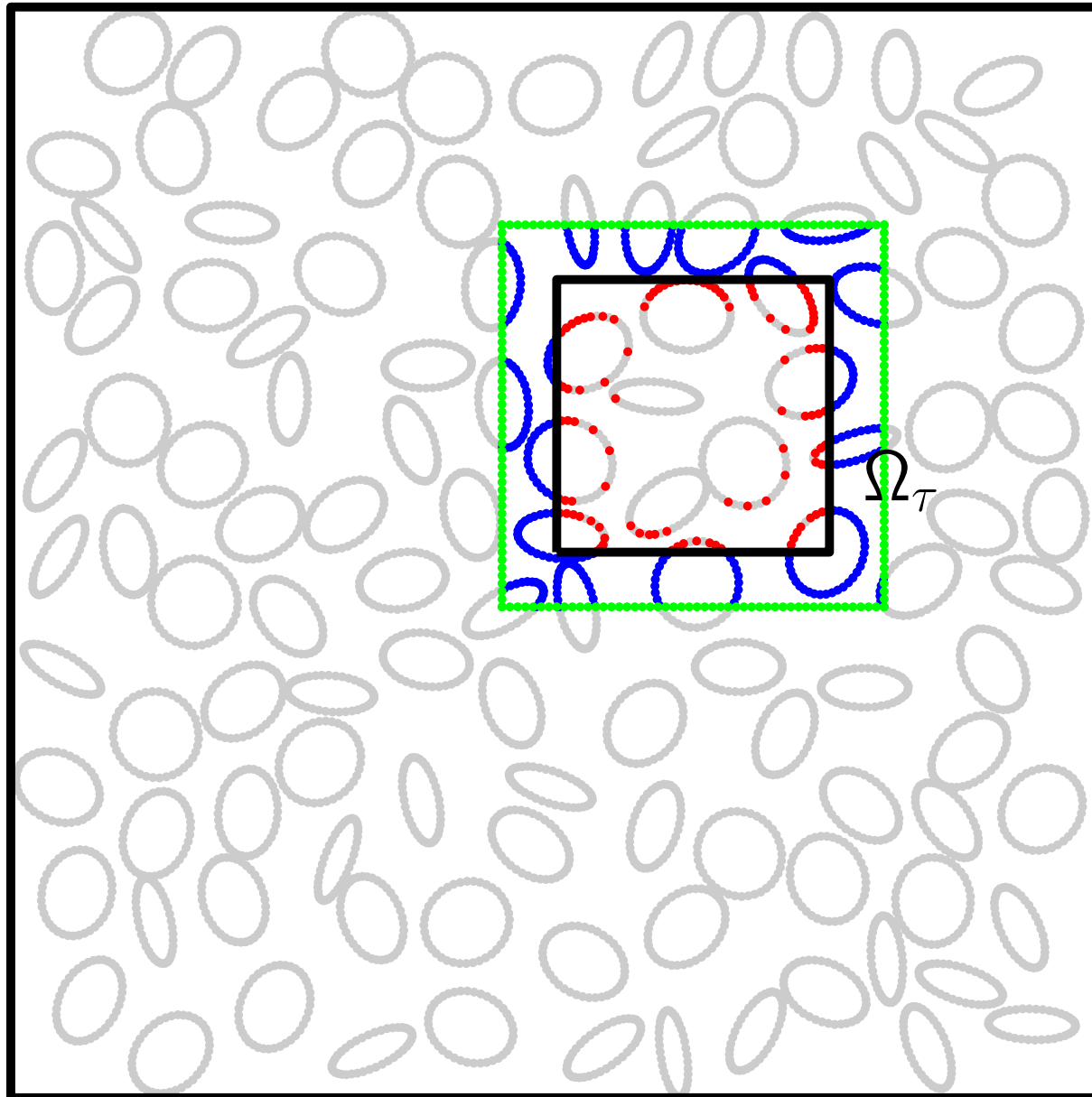


- Points in I_τ .
- Points in I_τ^{near} .
- Points in I_τ^{proxy} .

Constructing a data-sparse representation of an integral operator

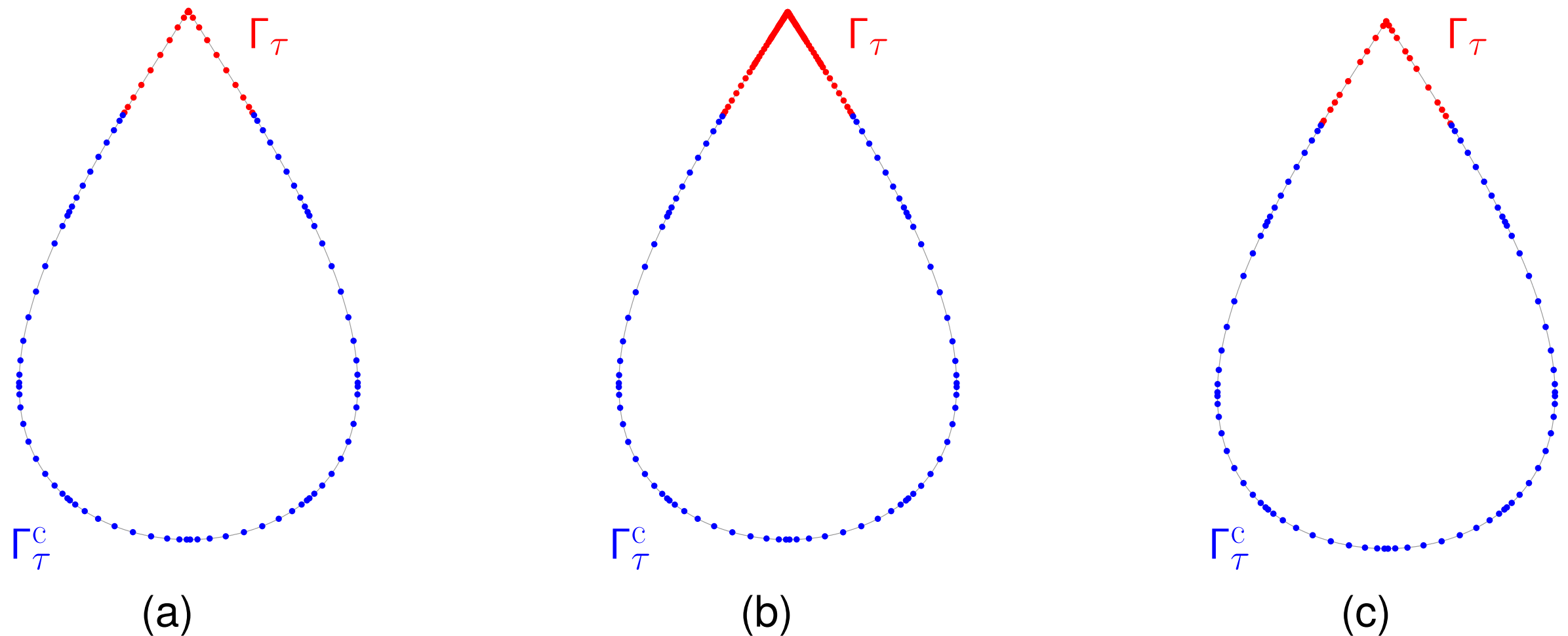
Illustration of the proxy point compression technique:

When compressing a subdomain I_τ (shown in red), we replace all “far-field” points (gray) by a set of artificial “proxy points” (green).



- Points in I_τ^{skel} .
- Points in I_τ^{near} .
- Points in I_τ^{proxy} .

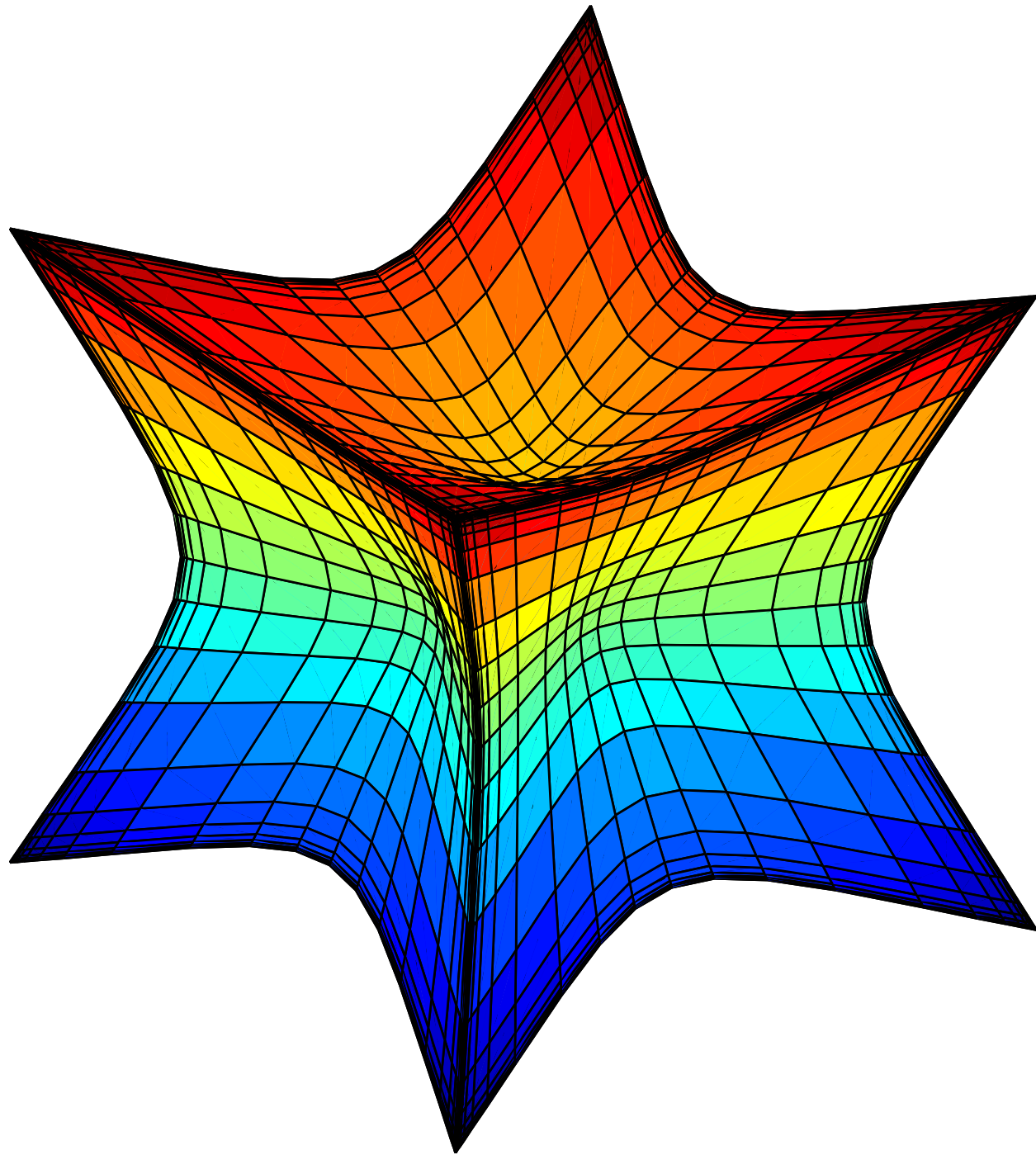
Consider a domain Γ discretized with Nyström based on a panel-based method using 10-point Gaussian quadrature. Suppose further that Γ has a corner, causing singularities in the layer potentials.



(a) Ignoring the singularity at the corner gives a small system but poor accuracy.

(b) Refining the grid near the corner gives great accuracy, but N gets very large.

(c) After local “compression” in which we compute a compressed scattering matrix for Γ_{τ} , we get the same accuracy as the discretization in (b), using as many nodes as in (a)!

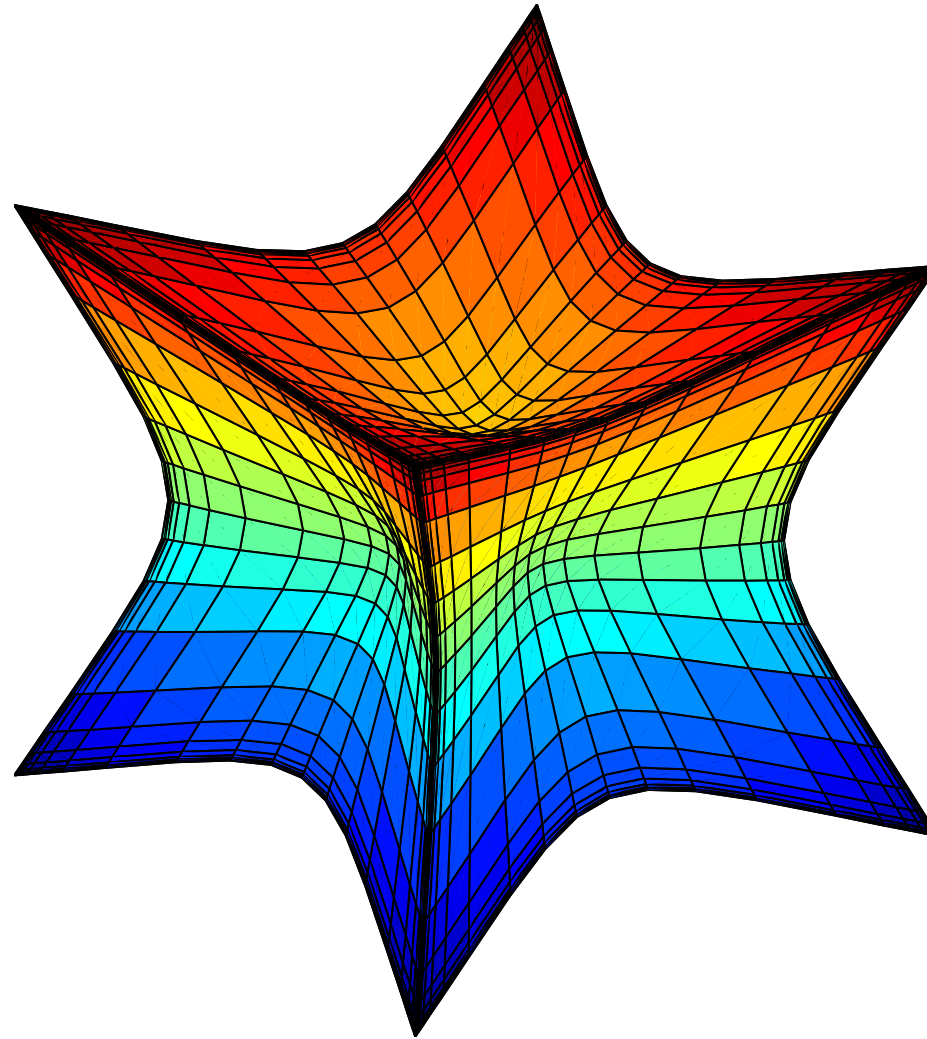


A surface Γ with corners and edges.

The grid has been refined to attain high accuracy.

Computing scattering matrices for the corners is conceptually easy (but laborious).

Numerical example — BIE on “edgy” surface



N_{tris}	N	E	T	$N_{\text{out}} \times N_{\text{in}}$
192	21 504	2.60×10^{-08}	$6.11 \times 10^{+02}$	617×712
432	48 384	2.13×10^{-09}	$1.65 \times 10^{+03}$	620×694
768	86 016	3.13×10^{-10}	$3.58 \times 10^{+03}$	612×685

Results from a Helmholtz problem (acoustic scattering) on the domain exterior to the “edgy” cube. The domain is about 3.5 wave-lengths in diameter.

Mesh refinement near edges & corners. The direct solver eliminates “extra” DOFs.

Fast direct solvers and high order methods:

“FEM-BEM coupling”

Consider the free space acoustic scattering problem

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

where

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

$$-\Delta u - \kappa^2 (1-b)u = -\kappa^2 b v$$

support(b)

Fast direct solvers and high order methods:

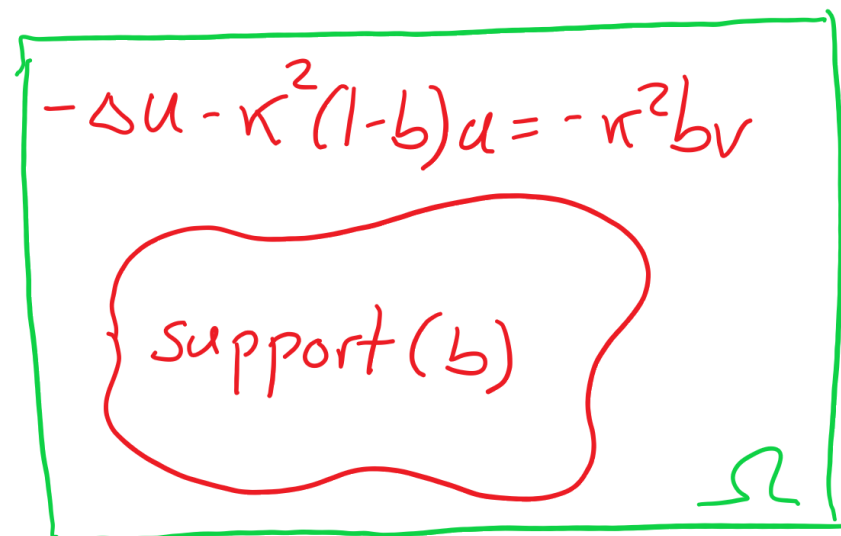
“FEM-BEM coupling”

Consider the free space acoustic scattering problem

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

where

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



$$-\Delta u - \kappa^2 u = 0 \text{ on } \Omega^c$$

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

Fast direct solvers and high order methods:

“FEM-BEM coupling”

Consider the free space acoustic scattering problem

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

where

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

$$-\Delta u - \kappa^2 (1 - b) u = -\kappa^2 b v$$

$$-\Delta u - \kappa^2 u = 0 \text{ on } \Omega^c$$

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

On Ω :

- Variable coefficient PDE.

On Ω^c :

- Constant coefficient PDE.

Fast direct solvers and high order methods:

“FEM-BEM coupling”

Consider the free space acoustic scattering problem

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

where

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

$-\Delta u - \kappa^2 (1-b)u = -\kappa^2 b v$

support(b)

$-\Delta u - \kappa^2 u = 0$ on Ω^c

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

On Ω :

- Variable coefficient PDE.
- Use HPS.

On Ω^c :

- Constant coefficient PDE.
- Use BIE.

Fast direct solvers and high order methods:

“FEM-BEM coupling”

Consider the free space acoustic scattering problem

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

where

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

$$-\Delta u - \kappa^2 (1 - b) u = -\kappa^2 b v$$

$$-\Delta u - \kappa^2 u = 0 \text{ on } \Omega^c$$

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

On Ω :

- Variable coefficient PDE.
- Use HPS.
- Build DtN for $\partial\Omega$.

On Ω^c :

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

Fast direct solvers and high order methods:

“FEM-BEM coupling”

Consider the free space acoustic scattering problem

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

where

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

$$-\Delta u - \kappa^2 (1 - b) u = -\kappa^2 b v$$

support(b)

$$-\Delta u - \kappa^2 u = 0 \text{ on } \Omega^c$$

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

On Ω :

- Variable coefficient PDE.
- Use HPS.
- Build DtN for $\partial\Omega$.

On Ω^c :

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

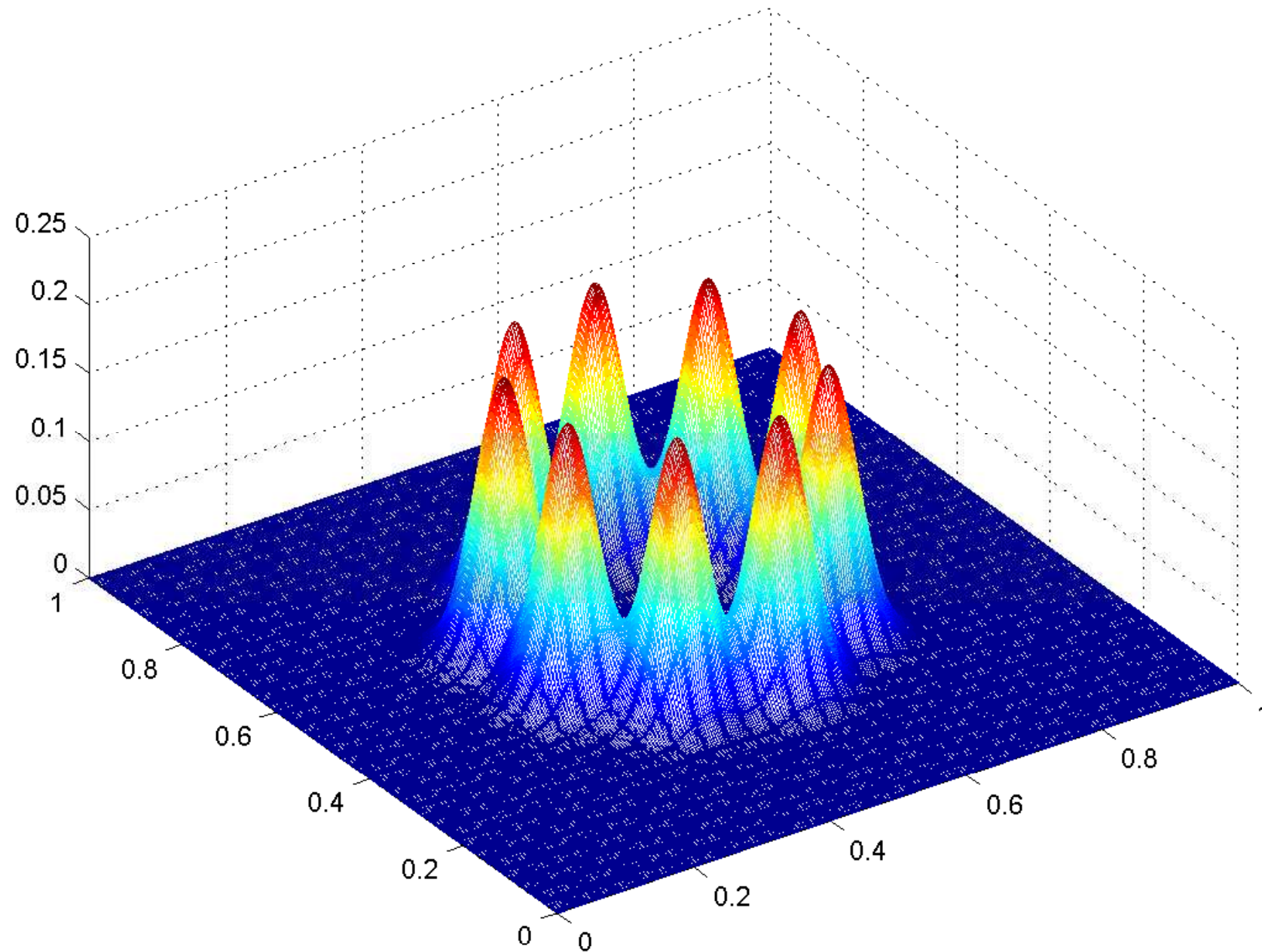
• Merge using fast operator algebra!

Fast direct solvers and high order methods:

“FEM-BEM coupling”

$$\begin{cases} -\Delta u_{\text{out}}(\mathbf{x}) - \kappa^2 (1 - b(\mathbf{x})) u_{\text{out}}(\mathbf{x}) = -\kappa^2 b(\mathbf{x}) u_{\text{in}}(\mathbf{x}) \\ \lim_{|\mathbf{x}| \rightarrow \infty} \sqrt{|\mathbf{x}|} (\partial_{|\mathbf{x}|} u_{\text{out}}(\mathbf{x}) - i\kappa u_{\text{out}}(\mathbf{x})) = 0 \end{cases}$$

The scattering potential b

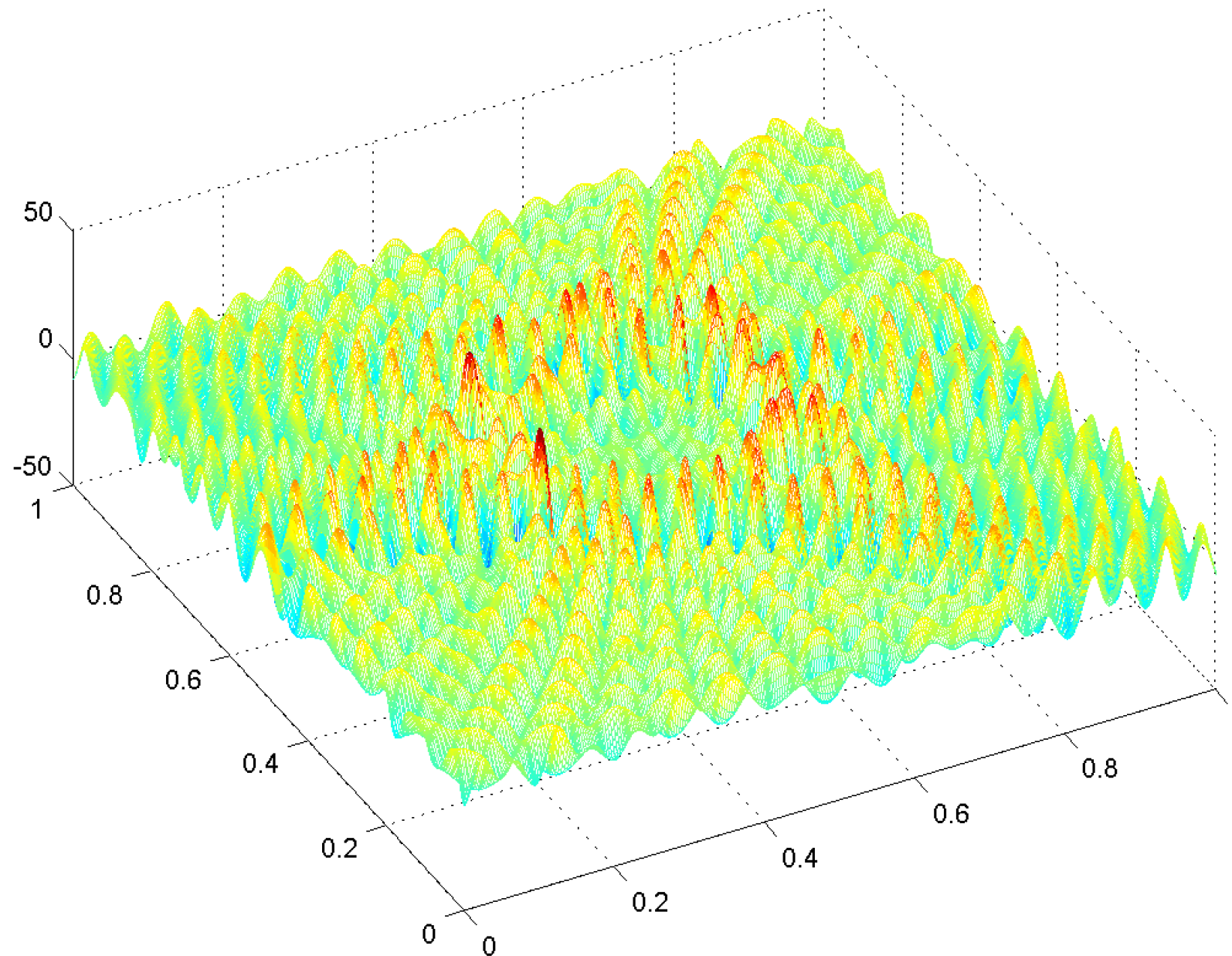


Fast direct solvers and high order methods:

“FEM-BEM coupling”

$$\begin{cases} -\Delta u_{\text{out}}(\mathbf{x}) - \kappa^2 (1 - b(\mathbf{x})) u_{\text{out}}(\mathbf{x}) = -\kappa^2 b(\mathbf{x}) u_{\text{in}}(\mathbf{x}) \\ \lim_{|\mathbf{x}| \rightarrow \infty} \sqrt{|\mathbf{x}|} (\partial_{|\mathbf{x}|} u_{\text{out}}(\mathbf{x}) - i\kappa u_{\text{out}}(\mathbf{x})) = 0 \end{cases}$$

The outgoing field u_{out} (resulting from an incoming plane wave $u_{\text{in}}(x) = \cos(\kappa x_1)$)



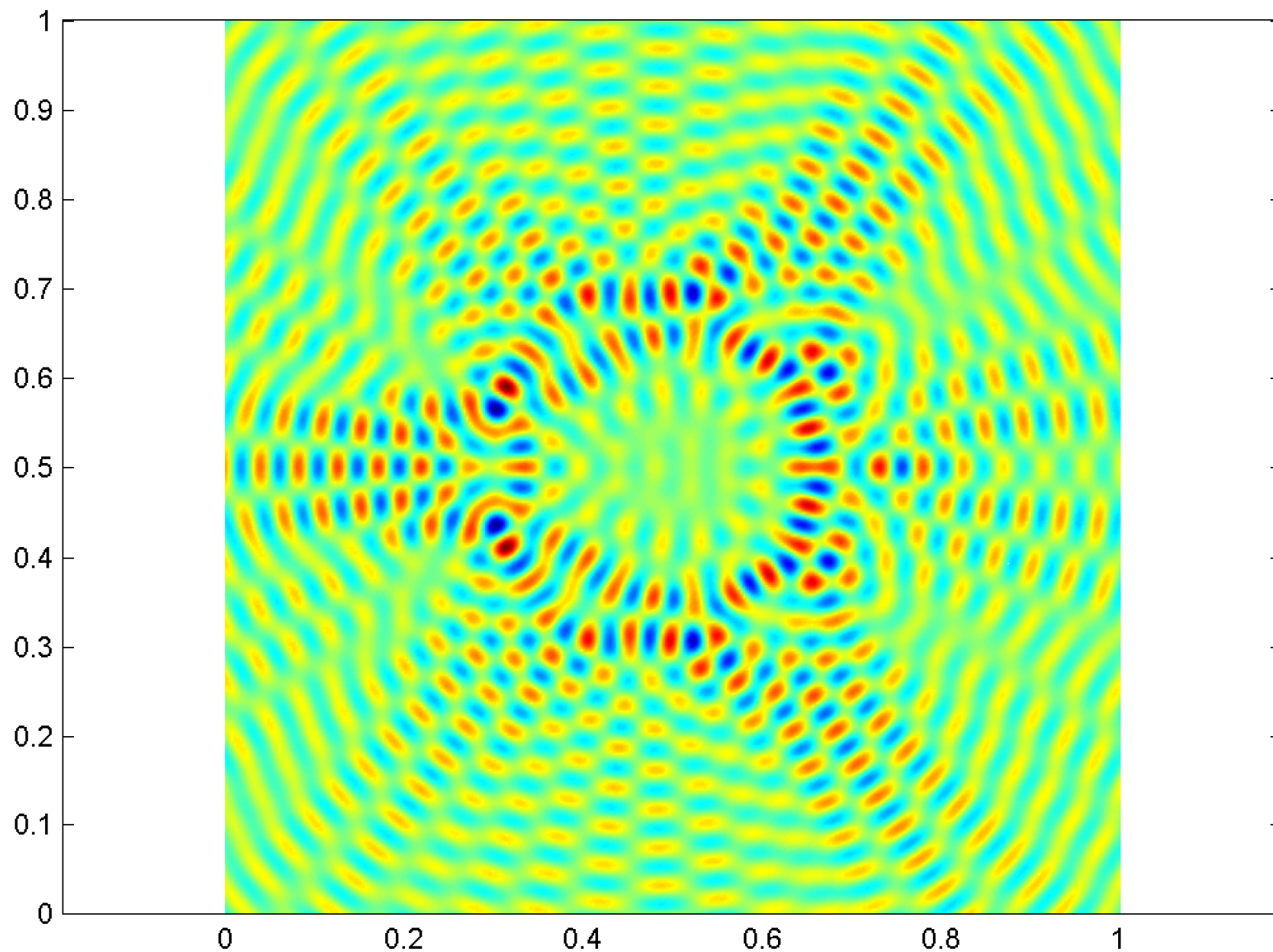
$N = 231\,361$ $T_{\text{build}} = 7.2 \text{ sec}$ $T_{\text{solve}} = 0.06 \text{ sec}$ $E \approx 10^{-7}$ (estimated)

Fast direct solvers and high order methods:

“FEM-BEM coupling”

$$\begin{cases} -\Delta u_{\text{out}}(\mathbf{x}) - \kappa^2 (1 - b(\mathbf{x})) u_{\text{out}}(\mathbf{x}) = -\kappa^2 b(\mathbf{x}) u_{\text{in}}(\mathbf{x}) \\ \lim_{|\mathbf{x}| \rightarrow \infty} \sqrt{|\mathbf{x}|} (\partial_{|\mathbf{x}|} u_{\text{out}}(\mathbf{x}) - i\kappa u_{\text{out}}(\mathbf{x})) = 0 \end{cases}$$

The outgoing field u_{out} (resulting from an incoming plane wave $u_{\text{in}}(x) = \cos(\kappa x_1)$)



$N = 231\,361$

$T_{\text{build}} = 7.2 \text{ sec}$

$T_{\text{solve}} = 0.06 \text{ sec}$

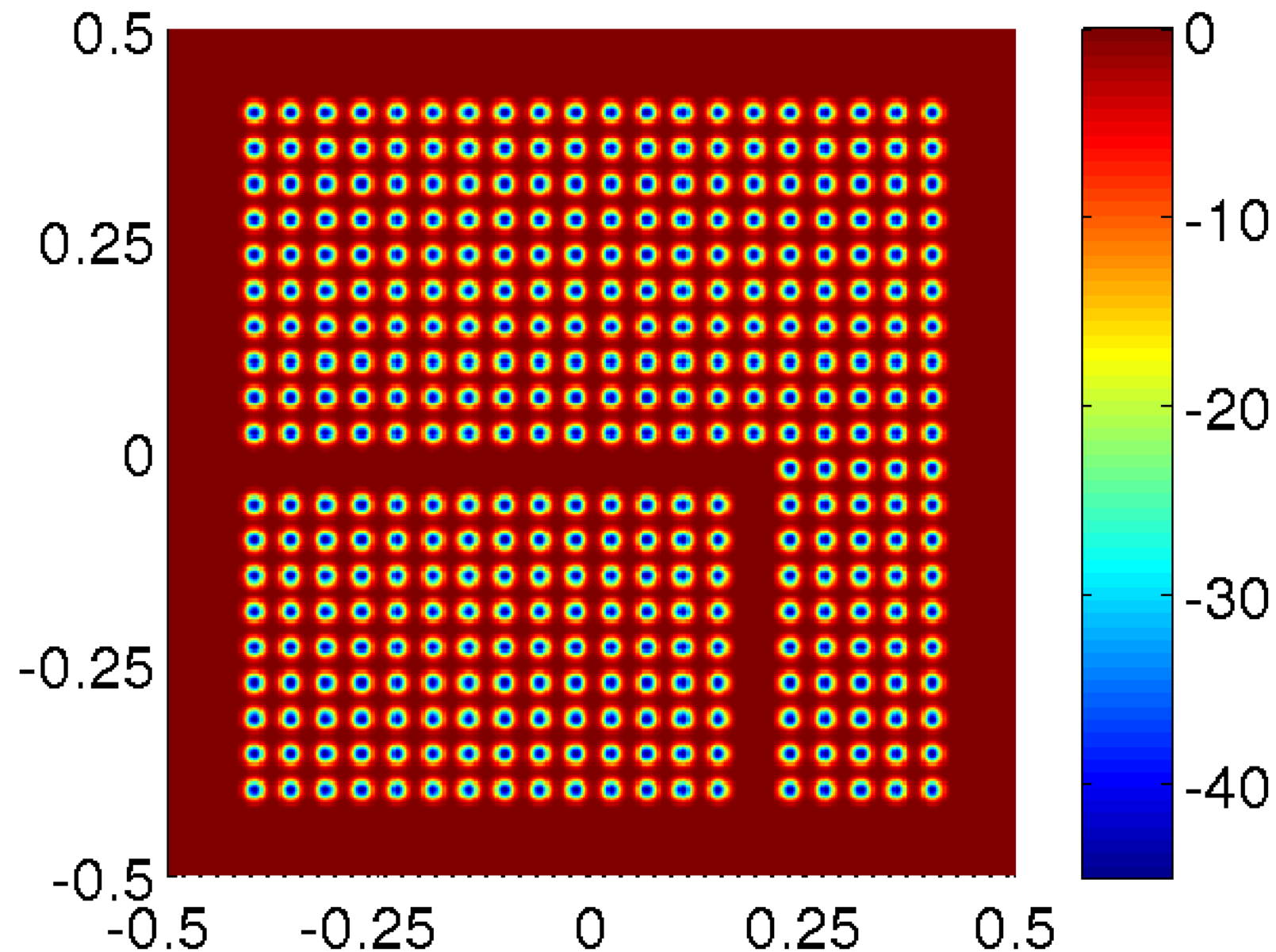
$E \approx 10^{-7}$ (estimated)

Fast direct solvers and high order methods:

“FEM-BEM coupling”

$$\begin{cases} -\Delta u_{\text{out}}(\mathbf{x}) - \kappa^2 (1 - b(\mathbf{x})) u_{\text{out}}(\mathbf{x}) = -\kappa^2 b(\mathbf{x}) u_{\text{in}}(\mathbf{x}) \\ \lim_{|\mathbf{x}| \rightarrow \infty} \sqrt{|\mathbf{x}|} (\partial_{|\mathbf{x}|} u_{\text{out}}(\mathbf{x}) - i\kappa u_{\text{out}}(\mathbf{x})) = 0 \end{cases}$$

The scattering potential b — now a photonic crystal with a wave guide.



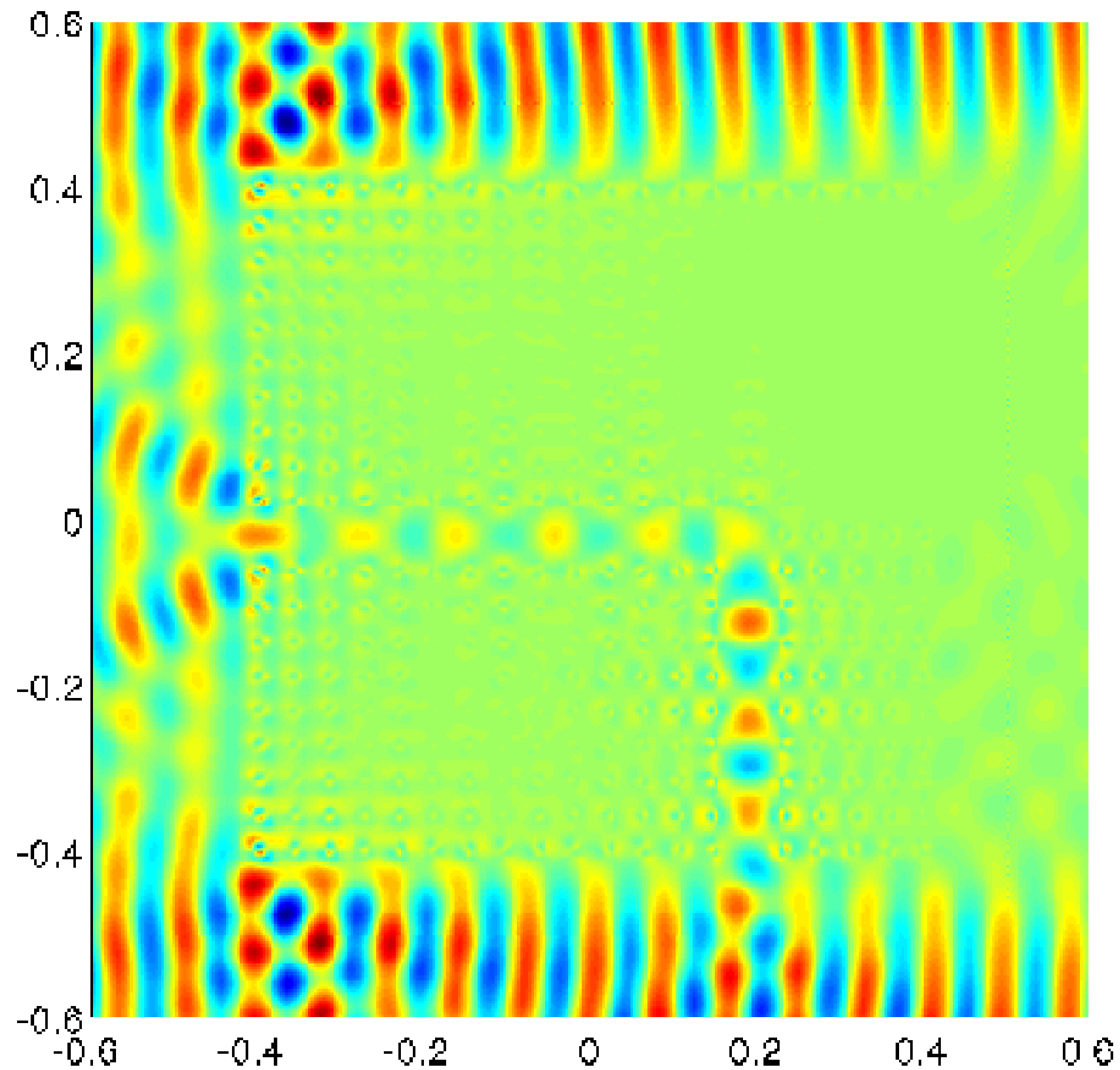
$N = 231\,361$ $T_{\text{build}} = 7.2 \text{ sec}$ $T_{\text{solve}} = 0.06 \text{ sec}$ $E \approx 10^{-6}$ (estimated)

Fast direct solvers and high order methods:

“FEM-BEM coupling”

$$\begin{cases} -\Delta u_{\text{out}}(\mathbf{x}) - \kappa^2 (1 - b(\mathbf{x})) u_{\text{out}}(\mathbf{x}) = -\kappa^2 b(\mathbf{x}) u_{\text{in}}(\mathbf{x}) \\ \lim_{|\mathbf{x}| \rightarrow \infty} \sqrt{|\mathbf{x}|} (\partial_{|\mathbf{x}|} u_{\text{out}}(\mathbf{x}) - i\kappa u_{\text{out}}(\mathbf{x})) = 0 \end{cases}$$

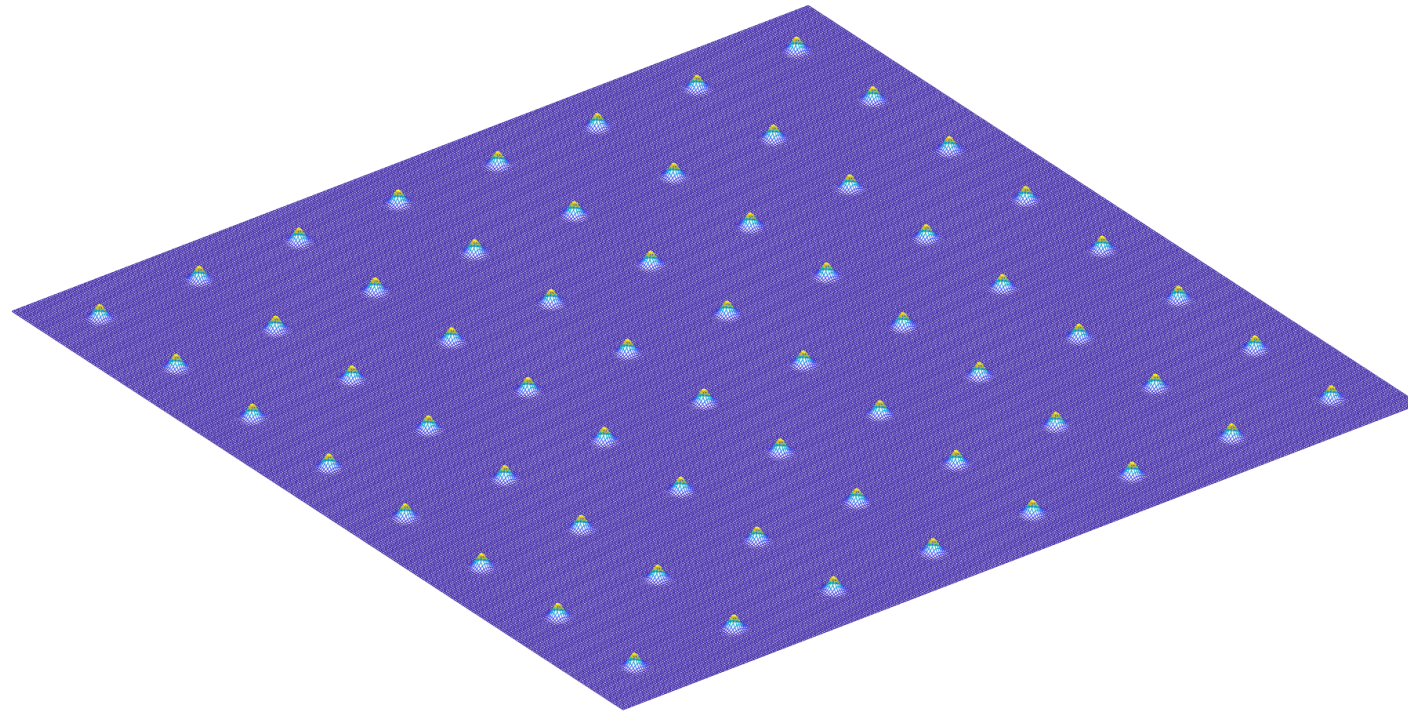
The total field $u = u_{\text{in}} + u_{\text{out}}$ (resulting from an incoming plane wave $u_{\text{in}}(x) = \cos(\kappa x_1)$).



Fast direct solvers and high order methods:

rough surface scattering

Consider acoustic scattering from an infinite half-plane with 8×8 bumps:



Each bump is 0.5λ tall and 2λ wide. Total domain is $44\lambda \times 44\lambda$.

The problem is formulated as a BIE, and is discretized using a “Zeta-corrected” quadrature rule. A direct solver with weak admissibility and $O(N^{1.5})$ scaling is used.

26M points total, $T_{\text{build}} = 1200\text{s}$, $T_{\text{solve}} = 184\text{s}$

About three digits of accuracy in the computed solution. (Tentative!)

Work in progress - with Abi Gopal and Bowei Wu.

Inversion of rank structured matrices with strong admissibility.

Wednesday!

Inversion of rank structured matrices with strong admissibility.

Wednesday!

Recall:

- Only “well-separated” blocks are compressed. This keeps ranks bounded.
- Strong adm. is used in the FMM & in Barnes-Hut. It is “default” for \mathcal{H} -matrices.

Example: Red blocks are represented as full rank.

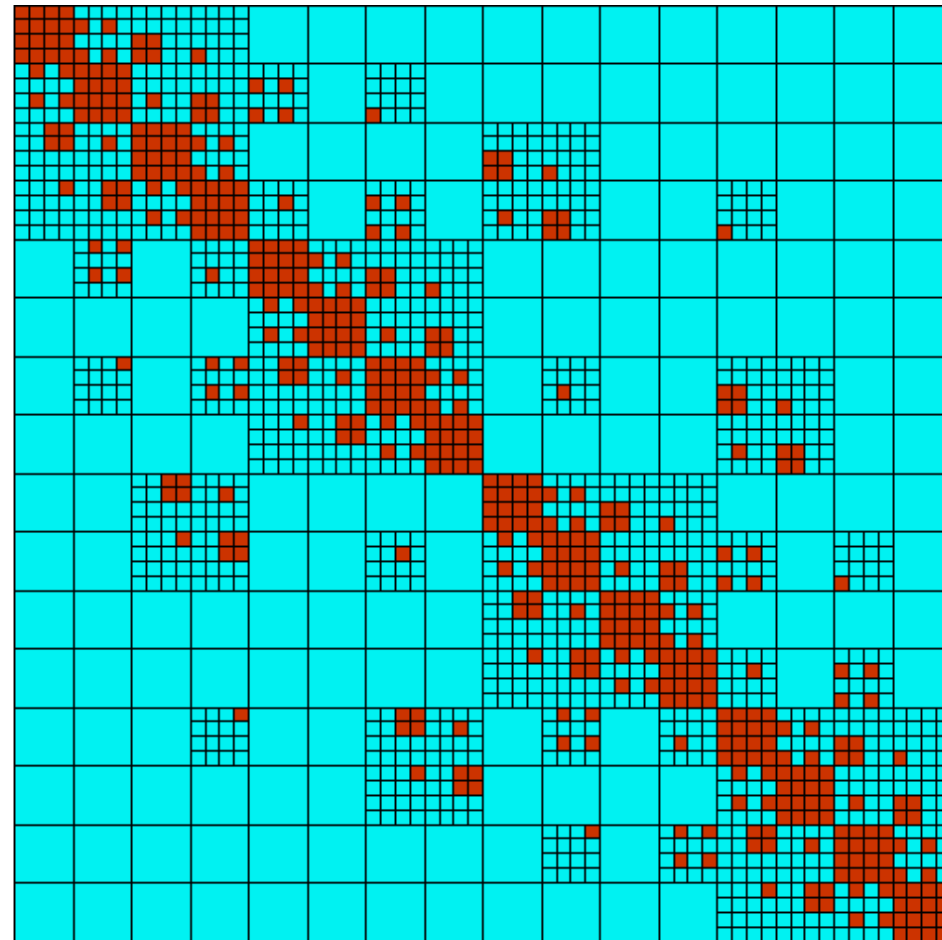


Image credit: Ambikasaran & Darve, arxiv.org #1407.1572

Inversion of rank structured matrices with strong admissibility.

Wednesday!

Recall:

- Only “well-separated” blocks are compressed. This keeps ranks bounded.
- Strong adm. is used in the FMM & in Barnes-Hut. It is “default” for \mathcal{H} -matrices.

Question: How do you invert a structure like this?

Inversion of rank structured matrices with strong admissibility.

Wednesday!

Recall:

- Only “well-separated” blocks are compressed. This keeps ranks bounded.
- Strong adm. is used in the FMM & in Barnes-Hut. It is “default” for \mathcal{H} -matrices.

Question: How do you invert a structure like this?

- LU factorization of \mathcal{H} -matrices. Key motivation for \mathcal{H} -matrix arithmetic. Hard to do efficiently. Ranks grow, so *recompression* is essential (inversion formulas are *not* exact). Some success as pre-conditioner.
- LU factorization of \mathcal{H}^2 -matrices. Even harder! Recent progress by Steffen Börm.
- Inverse FMM by Eric Darve. Simpler data structures. Memory hog!!
- *Strong recursive skeletonization* by Minden, Ho, Damle, and Ying. Multiscale Modeling & Simulation, **15**(2), 2017.

Very promising, but remains a memory hog.

Subject of current work in my group (see Wednesday talk):

- Development of HPC code that exploits GPU acceleration, and allows for distributed memory implementations. *With Chao Chen of North Carolina State University.*
- *[New!]* “Randomized strong recursive skeletonization”. Uses a *black box* randomized method to simultaneously compress and factorize an \mathcal{H}^2 -matrix. *With Anna Yesypenko.*

Coda: High order discretizations

Direct solvers are ideal for combining with *high order discretization*.

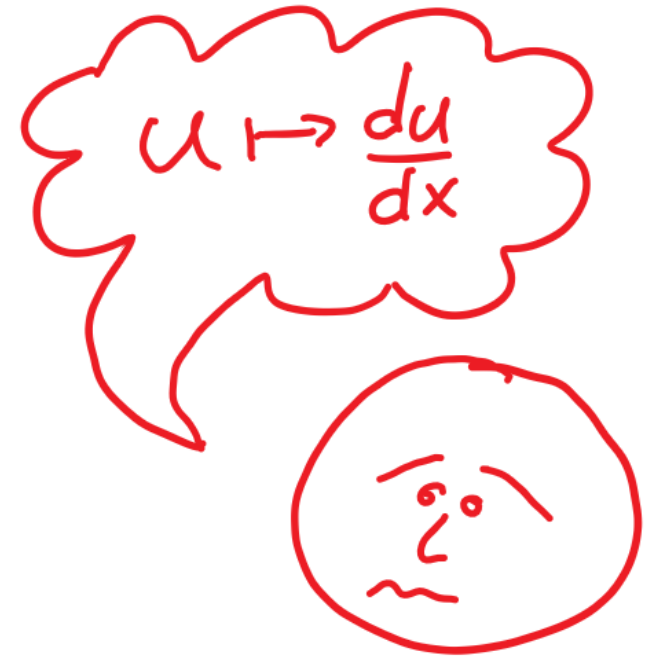
- Direct solvers use a lot of memory per degree of freedom.
→ *You want to maximize the oomph per DOF.*
- Direct solvers are particularly well suited for medium frequency wave problems.
→ *Need high accuracy due to ill-conditioned physics.*
- High order methods sometimes lead to more ill-conditioned systems.
→ *Can be hard to get iterative solvers to converge.*

When a direct solvers scores a clear win against iterative method, the use of high order discretizations often plays a central role.

Key Ideas:

The solution operator of a linear elliptic PDE is “friendly.”

- Smoothing.
- Stable.



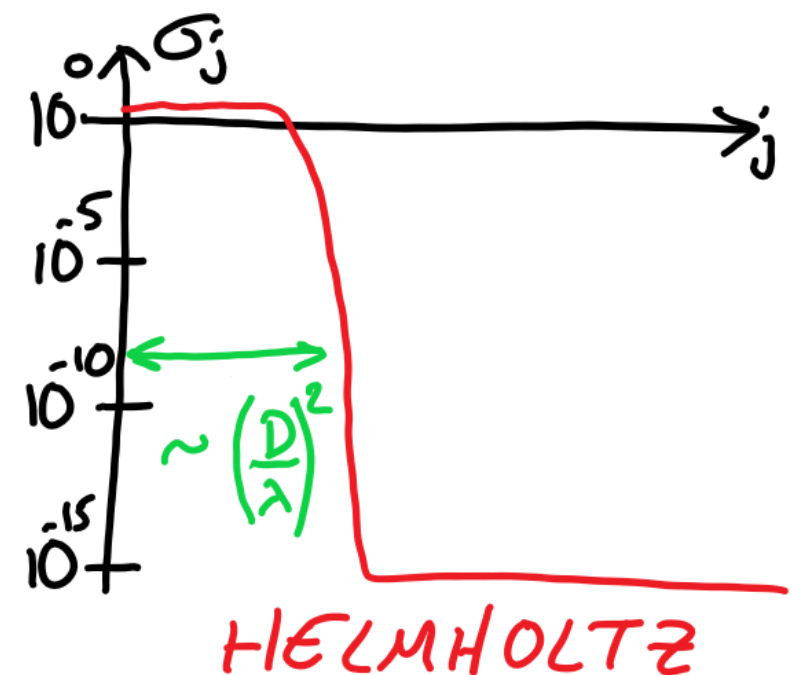
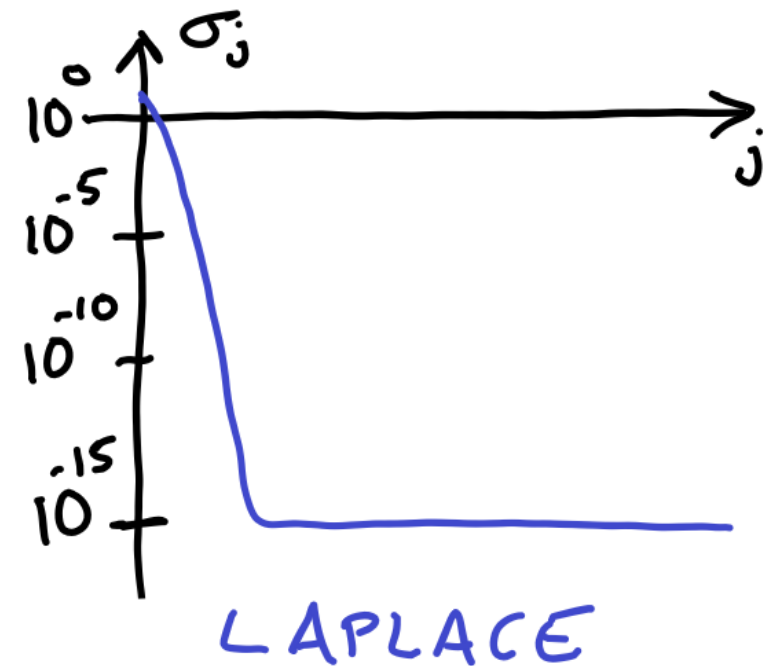
Key Ideas:

The solution operator of a linear elliptic PDE is “friendly.”

- Smoothing.
- Stable.

Long range interactions are low rank.

- Cf. St Venant principle, multipole expansions, etc.
- Smoothness is *not* necessary.
- Numerical compression is essential.
- Wave problems with small λ remain challenging.



Key Ideas:

The solution operator of a linear elliptic PDE is “friendly.”

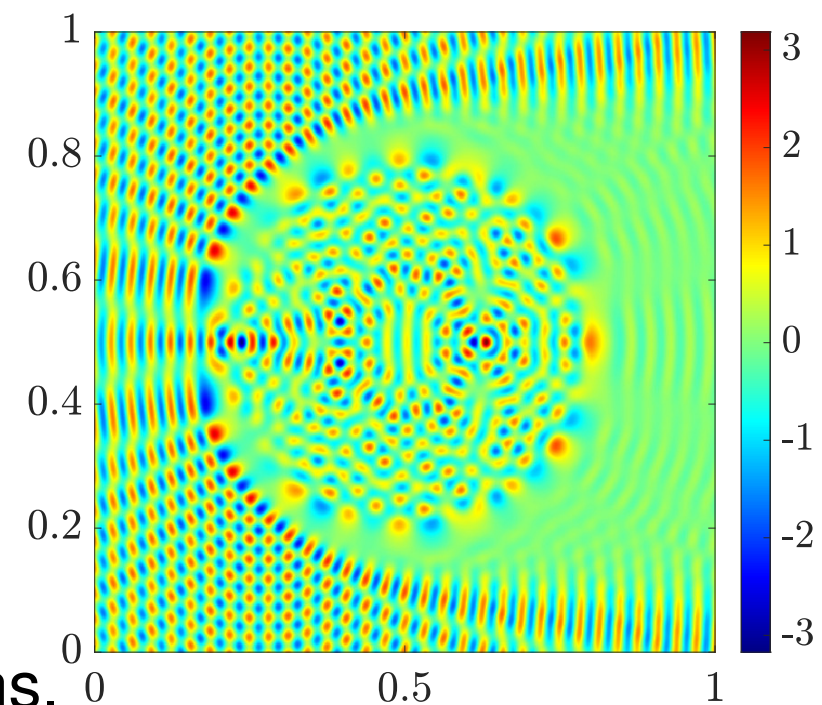
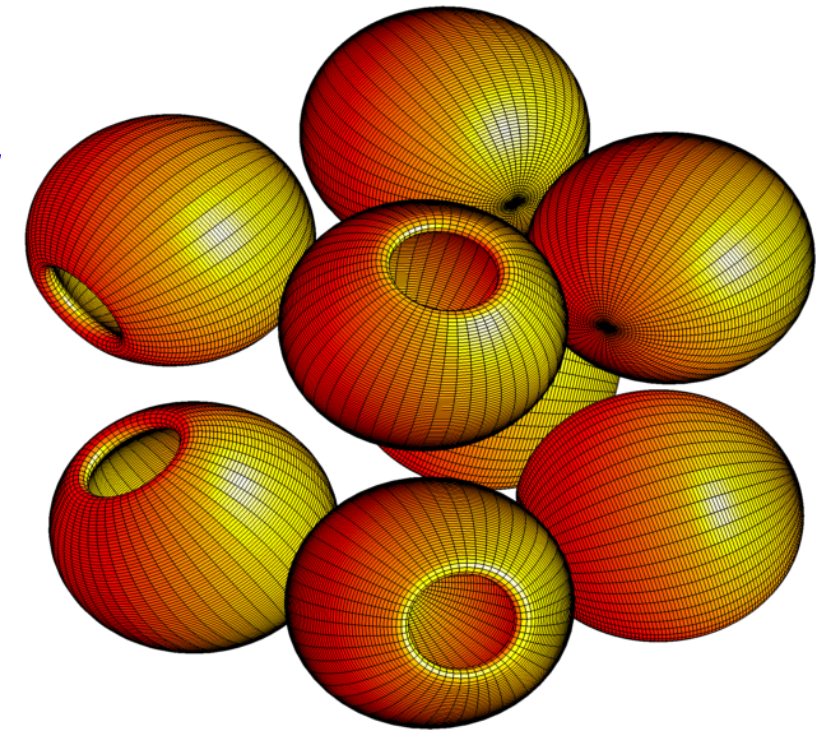
- Smoothing.
- Stable.

Long range interactions are low rank.

- Cf. St Venant principle, multipole expansions, etc.
- Smoothness is *not* necessary.
- Numerical compression is essential.
- Wave problems with small λ remain challenging.

High-order discretizations + FDS.

- Maximize the work done by each DOF to save memory.
- Perfect for ill-conditioned problems with oscillatory solutions.
- Requires care in choosing discretization scheme.



New randomized methods for matrix algebra → acceleration & simplification. Wednesday

Where we are now:

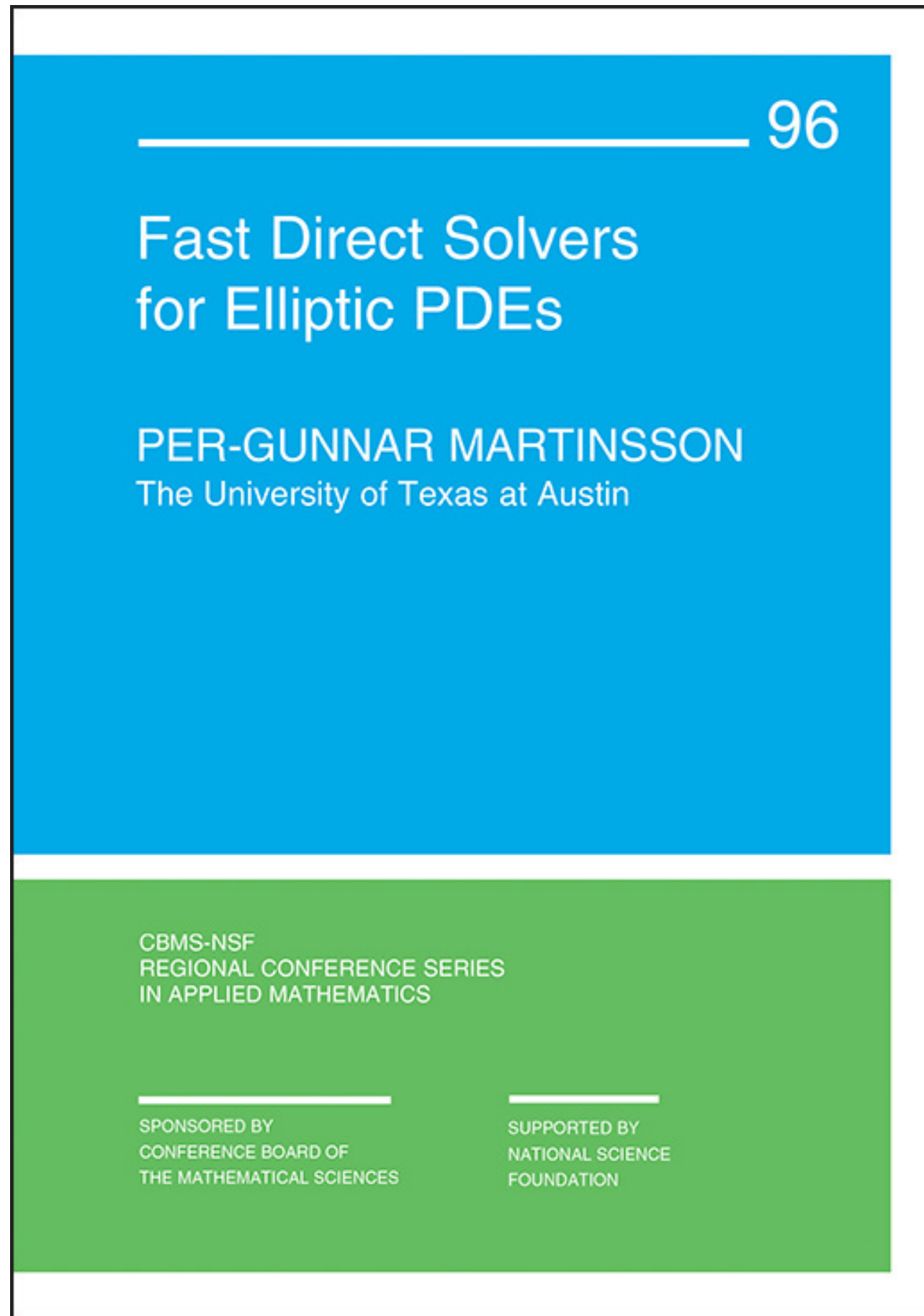
- We have developed direct solvers with $O(N)$ complexity for elliptic PDEs with non-oscillatory (or “mildly oscillatory”) solutions for most standard environments:
 - Sparse matrices from FEM/FD/composite spectral/... in both 2D and 3D.
 - Boundary integral equations in 2D and 3D. (Work in progress ...)
- Advantages of direct solvers:
 - Often instantaneous solves once a solution operator has been built.
 - Can eliminate problems with slow convergence of iterative solvers.
 - Enables “operator algebra”. (FEM-BEM coupling example.)
 - Communication efficient.
- Disadvantages of direct solvers:
 - Memory hogs. (But distributed memory is OK.)
 - The build stage is still slow for many 3D problems. (We are in the process of fixing this!)

Where to go next:

New powerful tool available → lots of opportunities!

- Explore happy couplings:
 - Direct solver + high order discretization. *(Helps with memory. Wave problems.)*
 - Direct solver + integral equation formulations. *(Need dense matrices anyway.)*
 - Direct solver + parallelization. *(Root of tree is cheap!)*
 - Direct solver + numerical coarse graining. *(Another talk...)*
- Parabolic and hyperbolic problems. Parallel-in-time methods?

More details in a 2019 monograph:



96

Fast Direct Solvers
for Elliptic PDEs

PER-GUNNAR MARTINSSON
The University of Texas at Austin

CBMS-NSF
REGIONAL CONFERENCE SERIES
IN APPLIED MATHEMATICS

SPONSORED BY
CONFERENCE BOARD OF
THE MATHEMATICAL SCIENCES

SUPPORTED BY
NATIONAL SCIENCE
FOUNDATION

**Quick introduction to randomized algorithms for
low rank approximation**

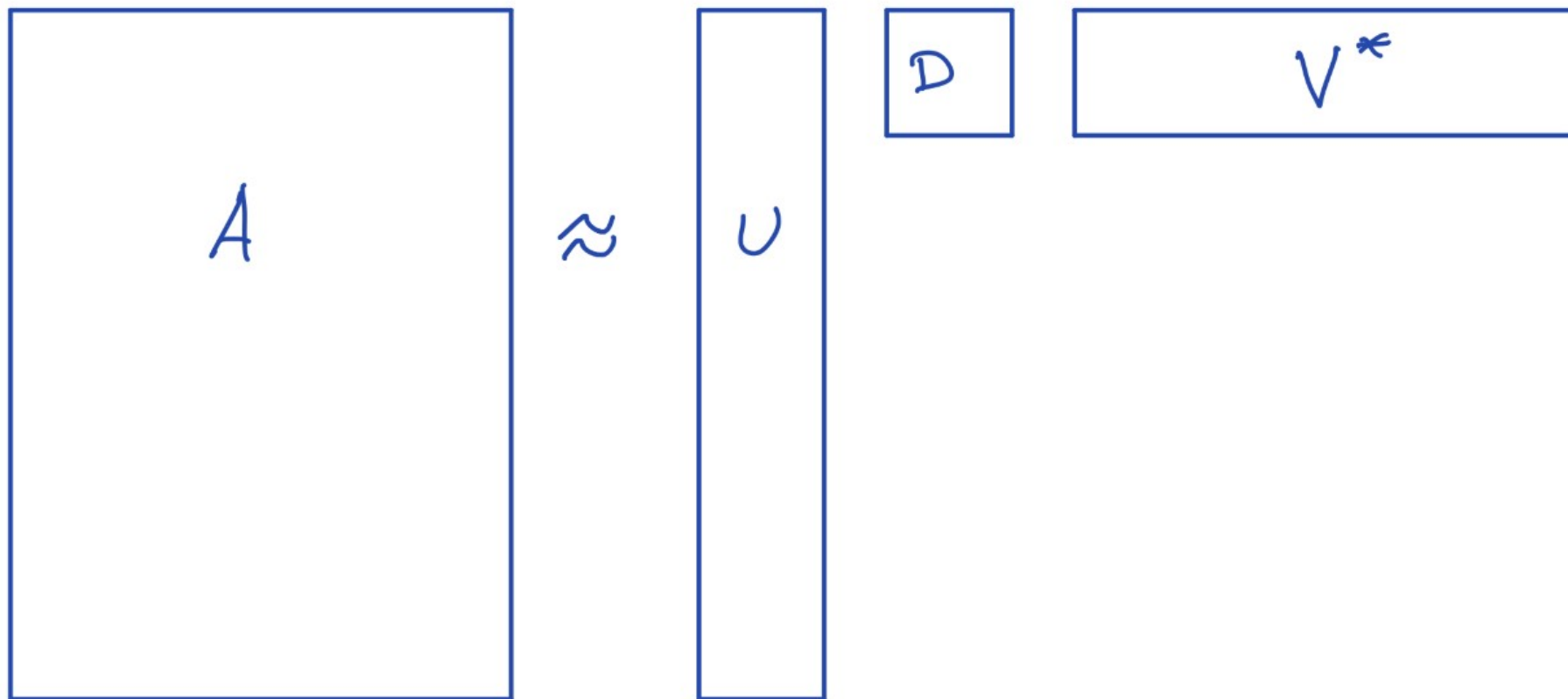
Randomized algorithms for low rank approximation:

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

$$\mathbf{A} \approx \mathbf{U} \mathbf{D} \mathbf{V}^*,$$

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

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



Randomized algorithms for low rank approximation:

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

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

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

Applications:

- Principal component analysis (fitting a hyperplane to data).
- Model reduction in analyzing physical systems.
- PageRank and other spectral methods in data analysis.
- **Compression of off-diagonal blocks in rank-structured matrices.**
- Diffusion geometry and manifold learning.
- Many, many more ...

Classical deterministic methods: Gram-Schmidt (column pivoted QR), power/subspace iteration, Krylov techniques, ...

Randomized algorithms for low rank approximation:

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

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

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

Solution:

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

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

Randomized algorithms for low rank approximation:

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

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

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

Solution:

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

Why does it work? When \mathbf{A} has exact rank k , the algorithm succeeds with probability 1. In the general case, it fails only if the columns of \mathbf{G} manage to all be close to orthogonal to a dominant right singular vector. *Very unlikely.*

Randomized algorithms for low rank approximation:

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

$$\mathbf{A} \approx \mathbf{U} \mathbf{D} \mathbf{V}^*,$$

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

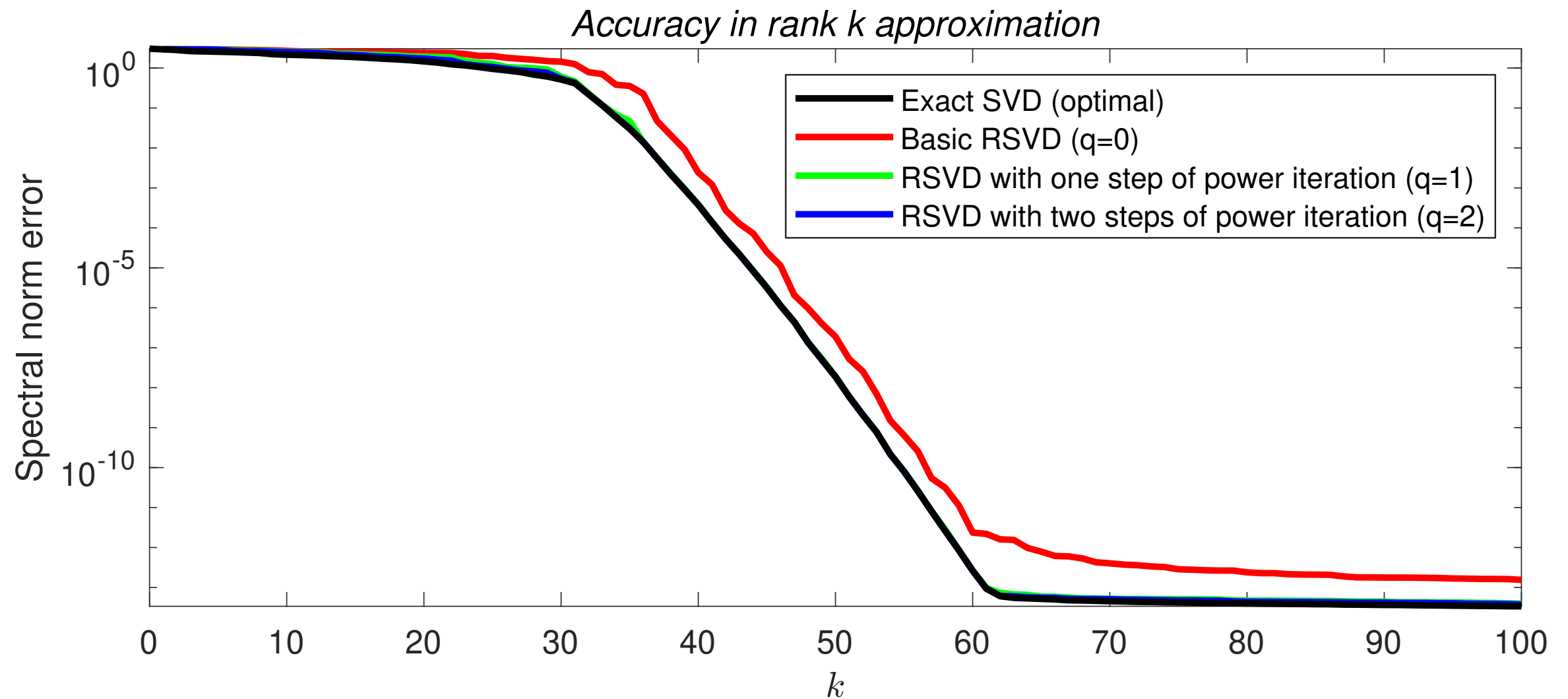
with \mathbf{U} and \mathbf{V} having orthonormal columns, and \mathbf{D} diagonal.

Solution:

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

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

Randomized low rank approximation:



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

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

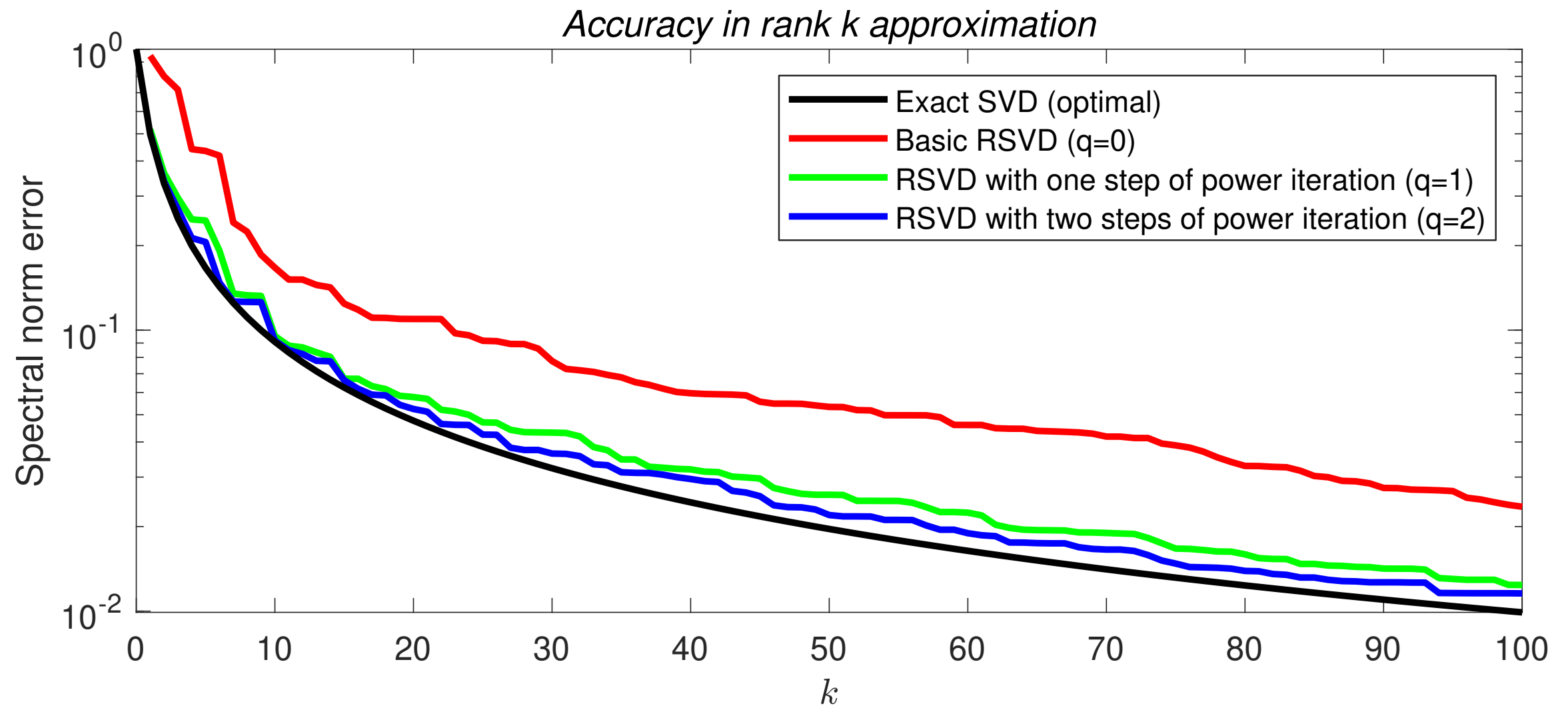
where \mathbf{P}_k is the orthogonal projection onto the first k columns of

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

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

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

Randomized low rank approximation:



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

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

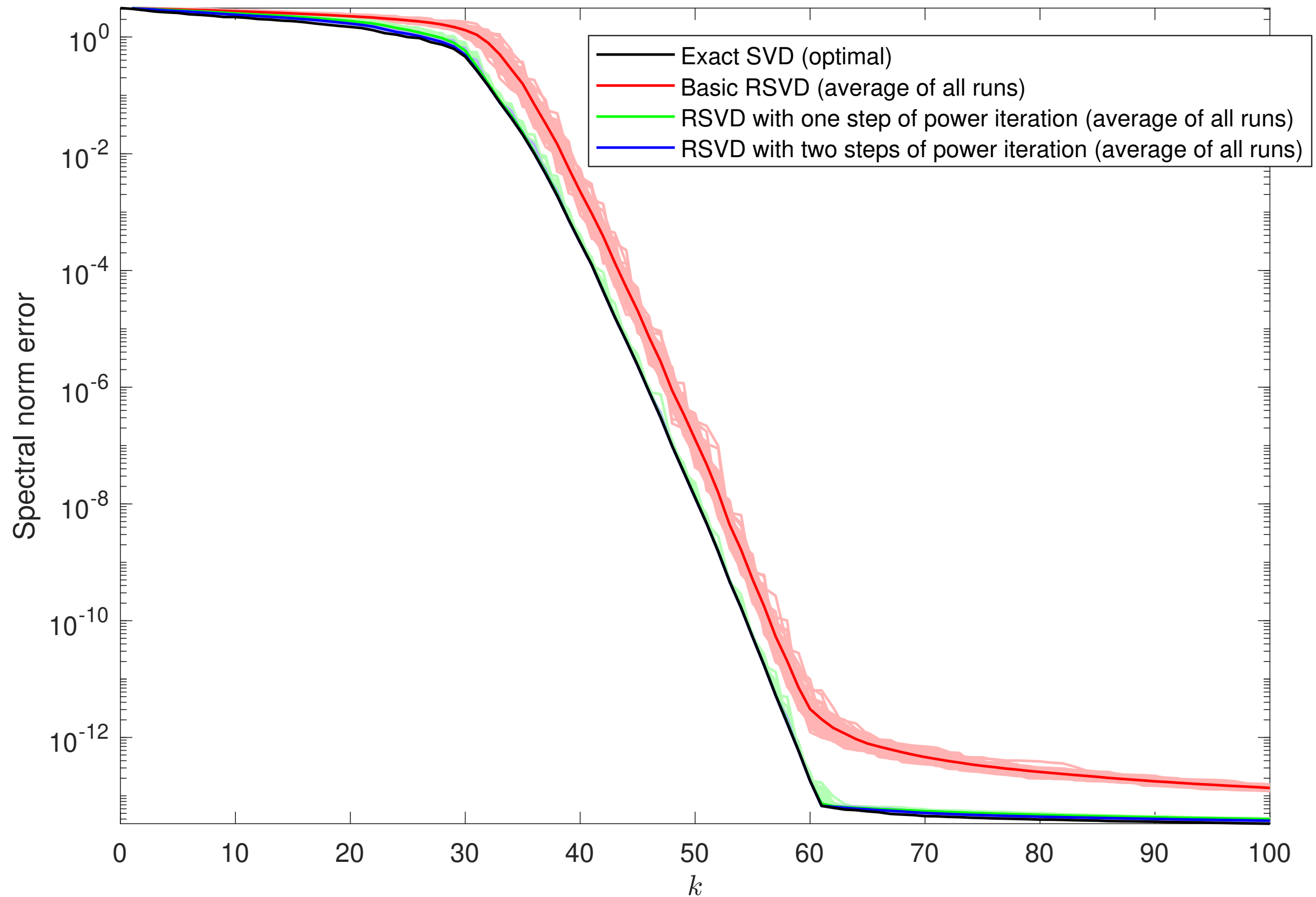
where \mathbf{P}_k is the orthogonal projection onto the first k columns of

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

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

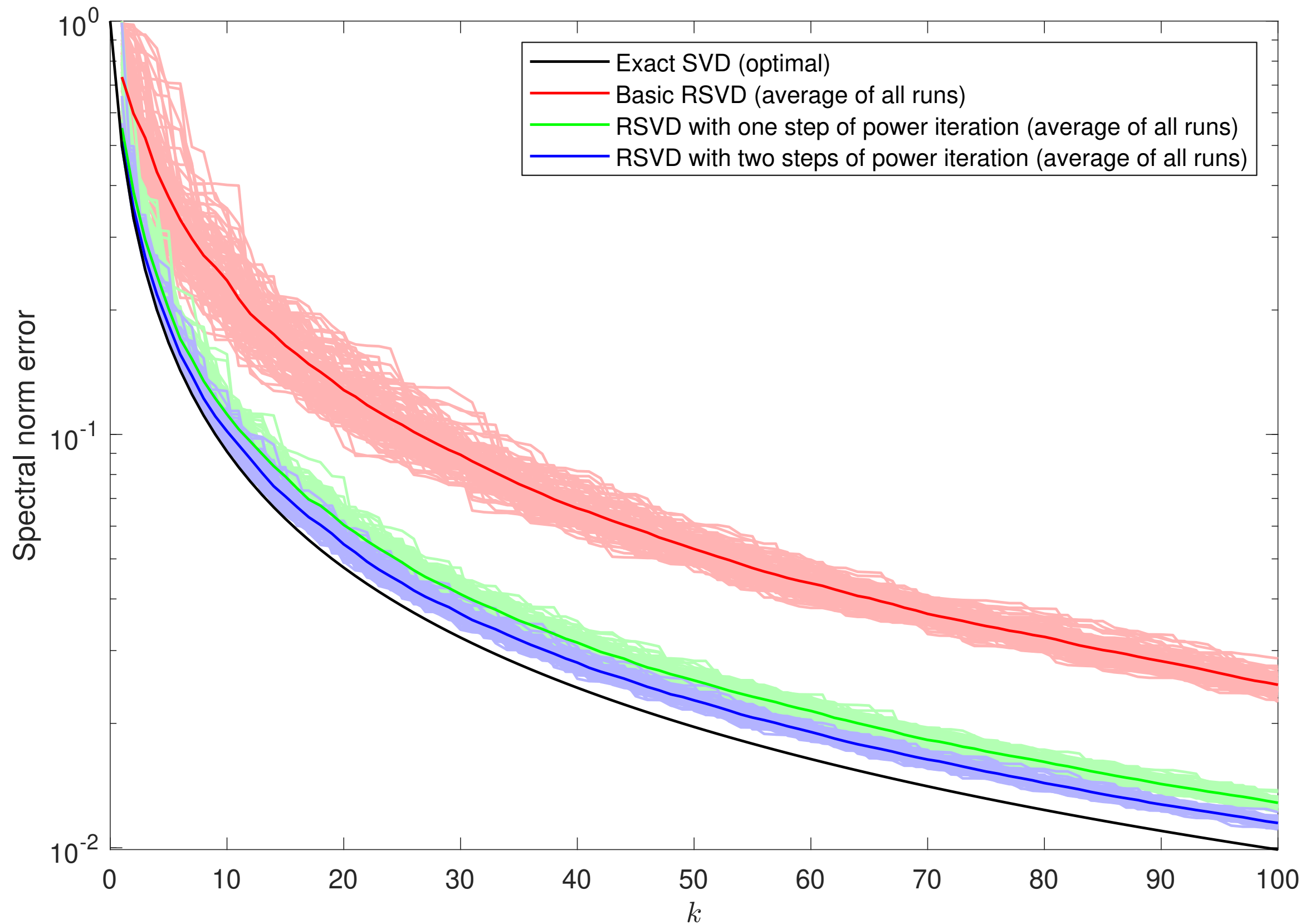
The matrix \mathbf{A} now has singular values that decay slowly.

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



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

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



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

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

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

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

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

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

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

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

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

Oversampling: By drawing a small number p of extra samples, we can prove that the error is close to theoretically minimal. Think $p = 5$ or $p = 10$.

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

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

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

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

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

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

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

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

Oversampling: By drawing a small number p of extra samples, we can prove that the error is close to theoretically minimal. Think $p = 5$ or $p = 10$. For instance, we have:

Theorem: [Halko, Martinsson, Tropp, 2009 & 2011] Let \mathbf{A} denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Let k denote a target rank and let p denote an over-sampling parameter. Let \mathbf{G} denote an $n \times (k + p)$ Gaussian matrix. Let \mathbf{Q} denote the $m \times (k + p)$ matrix $\mathbf{Q} = \text{orth}(\mathbf{AG})$. If $p \geq 2$, then

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

and

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

There are also bounds on the error when power iteration is used, the likelihood of large deviations from the expected value, and so on.

Focus of current work is construction of *à posteriori* error bounds, and estimates on the accuracy of computed singular *vectors*. (With Yijun Dong and Yuji Nakatsukasa.)

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

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

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

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

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

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

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

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

Key results on randomized SVD:

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

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

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

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

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

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

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

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

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

Key results on randomized SVD:

- 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).

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

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

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

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

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

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

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

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

Key results on randomized SVD:

- 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*.

Practical acceleration is achieved at ordinary matrix sizes.

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

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

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

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

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

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

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

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

Key results on randomized SVD:

- 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!

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

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

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

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

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

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

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

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

Key results on randomized SVD:

- 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*.

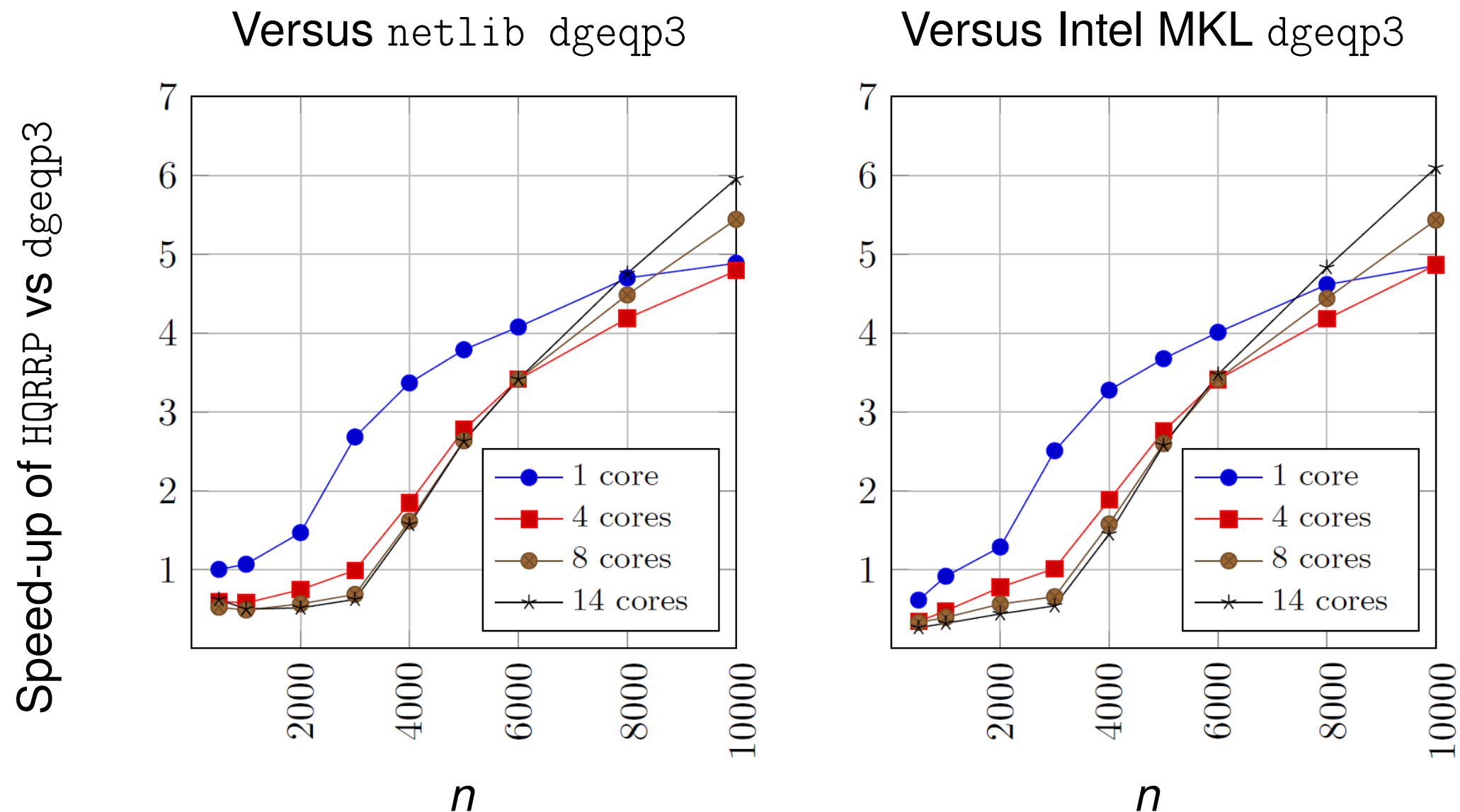
The idea is that you are allowed to observe each matrix element only once.

You cannot store the matrix.

Not possible with deterministic methods!

- The randomization idea can be used to overcome a classical problem in numerical linear algebra: How to efficiently implement column-pivoted QR factorizations. (How to cast the computation as BLAS3 operations instead of BLAS2.)

A very fast *randomized* implementation of column pivoted QR

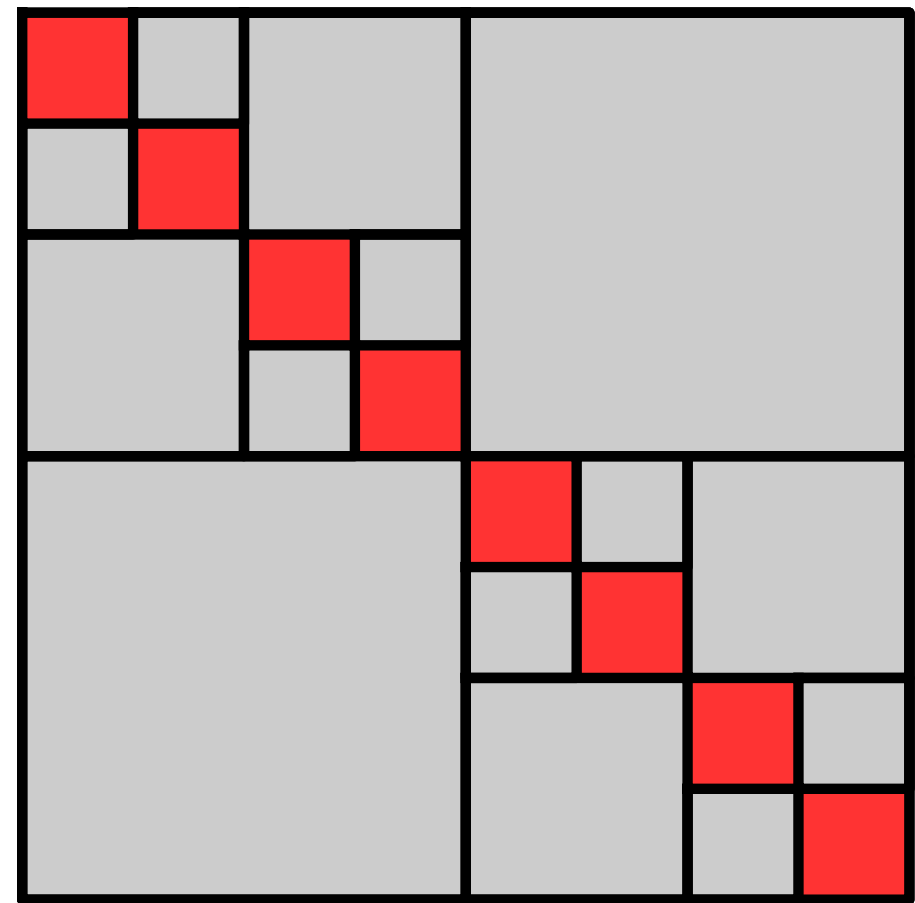


Speedup attained by a randomized algorithm for computing a full column pivoted QR factorization of an $n \times n$ matrix. The speed-up is measured versus LAPACK's faster routine `dgeqp3` as implemented in Netlib (left) and Intel's MKL (right). Our implementation was done in C, and was executed on an Intel Xeon E5-2695. Joint work with G. Quintana-Ortí, N. Heavner, and R. van de Geijn (SISC 2017). Closely related work by Duersch and Gu, SISC 2017 / SIREV 2020.

Rank structured matrices

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

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



All gray blocks have low rank.

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

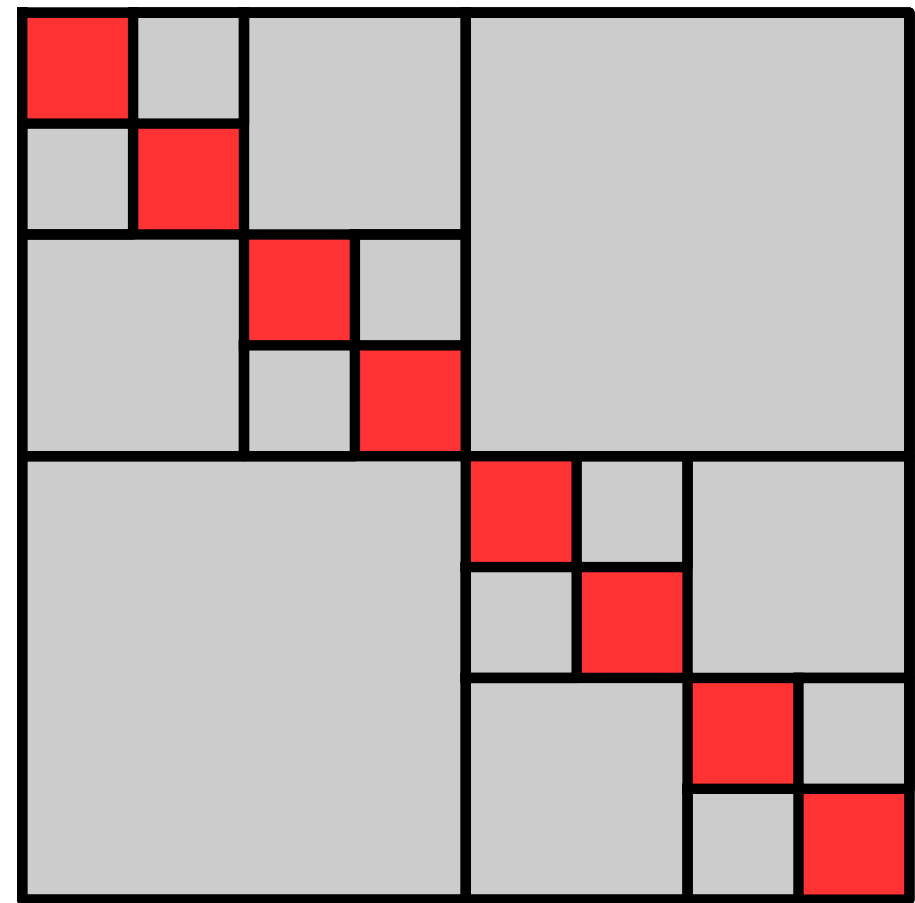
Ubiquitous applications in scientific computing: *Solution operators for elliptic PDEs, DtN operators, scattering matrices, Schur complements in sparse direct solvers, etc.*

More recently, have been shown to arise in data science as well — kernel matrices, covariance matrices, Hessians, etc.

Rank structured matrices

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

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



All gray blocks have low rank.

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

Ubiquitous applications in scientific computing: *Solution operators for elliptic PDEs, DtN operators, scattering matrices, Schur complements in sparse direct solvers, etc.*

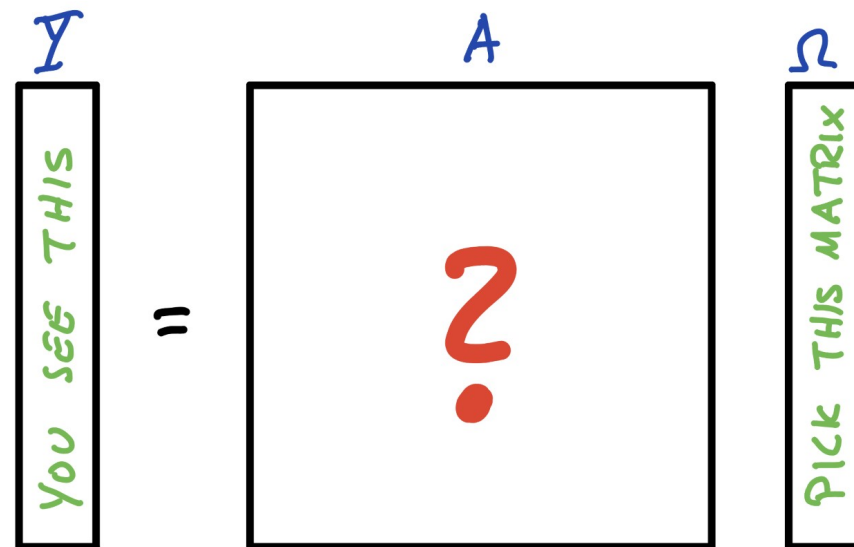
References: Fast Multipole Method (Greengard, Rokhlin); Panel Clustering (Hackbusch); \mathcal{H} - and \mathcal{H}^2 -matrices (Hackbusch et al); Hierarchically Block Separable matrices; Hierarchically Semi Separable matrices (Xia et al); HODLR matrices (Darve et al); BLR matrices (Buttari, Amestoy,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

Approximation of rank structured matrices

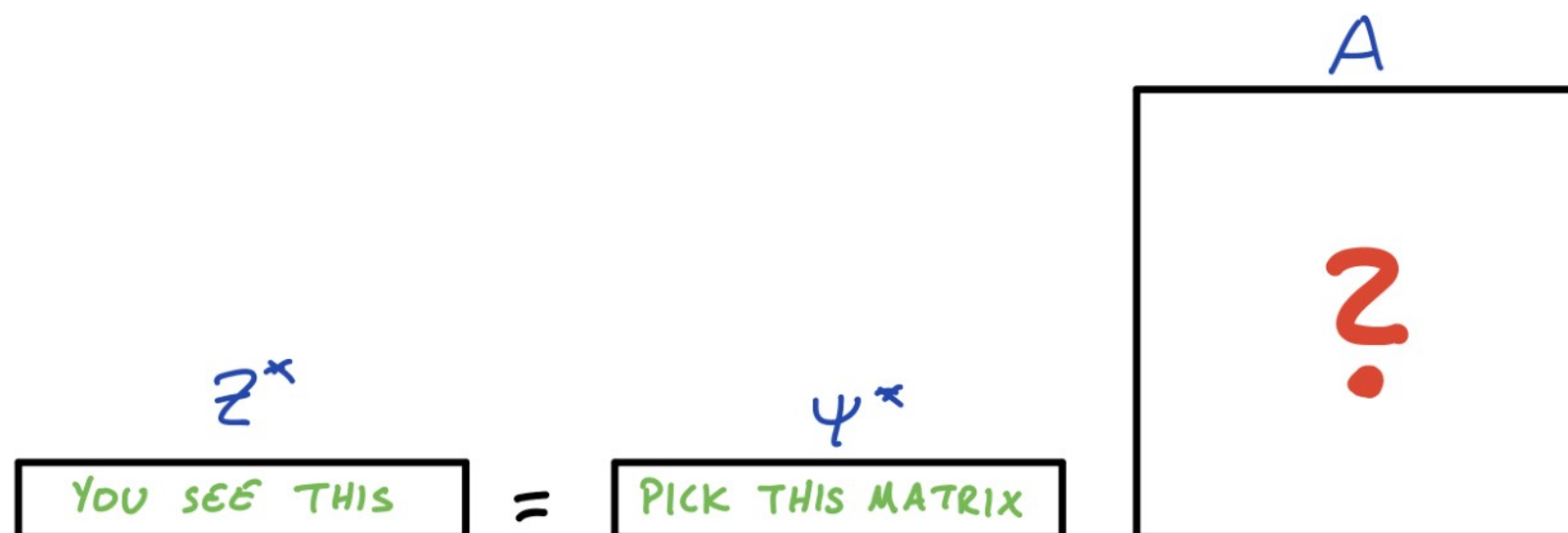
Environment: We are given a rank structured matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ (to be precise, \mathbf{A} is HBS/HSS of rank k). We assume that we can evaluate $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$ fast.

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

Sample the column space of the matrix:



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



Approximation of rank structured matrices

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

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

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

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

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

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

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

Approximation of rank structured matrices

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

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

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.
- Multiplication of operators. Useful for forming Dirichlet-to-Neumann operators, for combining solvers of multi-physics problems, etc.
- Compression of Schur complements that arise in the LU or Cholesky factorization of sparse matrices. This lets us overcome key bottlenecks (e.g. LU factorization of a “finite element” matrix is accelerated from $O(N^2)$ to close to linear complexity.)

Approximation of rank structured matrices

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

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

Available techniques for the rank structured case:

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

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

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

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

Approximation of rank structured matrices

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

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

Available techniques for the rank structured case:

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

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

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

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

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

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