Fast direct solvers for elliptic PDEs

Gunnar Martinsson
The University of Colorado at Boulder

Students:

Adrianna Gillman (now at Dartmouth)

Nathan Halko

Sijia Hao

Patrick Young (now at GeoEye Inc.)

Collaborators:

Eric Michielssen (Michigan)

Eduardo Corona (NYU)

Vladimir Rokhlin (Yale)

Mark Tygert (NYU)

Denis Zorin (NYU)

The talk will describe "fast direct" techniques for solving the linear systems arising from the discretization of linear boundary value problems (BVPs) of the form

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

where Ω is a domain in $\mathbb{R}^2$ or $\mathbb{R}^3$ with boundary Γ, and where A is an elliptic differential operator. Examples include:

- 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).

Example: Poisson equation with Dirichlet boundary data:

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

Discretization of linear Boundary Value Problems

↙ ↘

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

↓

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

↓

Fast solvers:
iterative (multigrid), $O(N)$,
direct (nested dissection), $O(N^{3/2})$.

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.

↓

Iterative solver accelerated by fast matrix-vector multiplier, $O(N)$.

Discretization of linear Boundary Value Problems

↙ ↘

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

↓

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

↓

Fast solvers:
iterative (multigrid), $O(N)$,
direct (nested dissection), $O(N^{3/2})$.
$O(N)$ direct solvers.

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.

↓

Iterative solver accelerated by fast matrix-vector multiplier, $O(N)$.
$O(N)$ direct solvers.

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

$$\mathbf{A}\,\mathbf{u} = \mathbf{b}, \tag{2}$$

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

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

Then an approximate solution to (2) is obtained by simply evaluating

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

The matrix $\mathbf{T}$ is typically constructed in a *compressed format* that allows the matrix-vector product $\mathbf{T}\,\mathbf{b}$ 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.*)

"Iterative" versus "direct" solvers

Two classes of methods for solving an $N \times N$ linear algebraic system

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

Iterative methods:

Examples: GMRES, conjugate gradients, Gauss-Seidel, *etc.*

Construct a sequence of vectors $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \ldots$ that (hopefully!) converge to the exact solution.

Many iterative methods access $\mathbf{A}$ only via its action on vectors.

Often require problem specific preconditioners.

High performance when they work well.
$O(N)$ solvers.

Direct methods:

Examples: Gaussian elimination,
LU factorizations, matrix inversion, *etc.*

Always give an answer. Deterministic.

Robust. No convergence analysis.

Great for multiple right hand sides.

Have often been considered too slow for high performance computing.

(Directly access elements or blocks of $\mathbf{A}$.)

(Exact except for rounding errors.)

Advantages of direct solvers over iterative solvers:

1. Applications that require a very large number of solves:
   - 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.

2. Problems that are relatively ill-conditioned:
   - 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.

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

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.

How do you construct direct solvers with less than $O(N^3)$ complexity?

For *sparse* matrices, algorithms such as "nested dissection" achieve $O(N^{1.5})$ or $O(N^2)$ complexity by reordering the matrix.

More recently, methods that exploit *data-sparsity* have achieved linear or close to linear complexity in a broad variety of environments.

In this talk, the *data-sparse* matrices under consideration have off-diagonal blocks that can to high accuracy (say ten of fifteen digits) be approximated by low-rank matrices.

Fast direct solvers for elliptic PDEs based on data-sparsity:

(Apologies to co-workers: A. Gillman, L. Greengard, D. Gueyffier, V. Rokhlin, M. Tygert, P. Young, ...)

1991 Data-sparse matrix algebra / wavelets: *Beylkin, Coifman, Rokhlin, et al*

1993 Fast inversion of 1D operators: *V. Rokhlin and P. Starr*

1996 scattering problems: *E. Michielssen, A. Boag and W.C. Chew,*

1998 factorization of non-standard forms: *G. Beylkin, J. Dunn, D. Gines,*

1998 $\mathcal{H}$-matrix methods: ***W. Hackbusch***, *B. Khoromskijet, S. Sauter, ...,*

2000 Cross approximation, matrix skeletons: etc., *E. Tyrtyshnikov*

2002 $O(N^{3/2})$ inversion of Lippmann-Schwinger equations: *Y. Chen,*

2002 "Hierarchically Semi-Separable" matrices: *M. Gu, S. Chandrasekharan.*

2002 (1999?) $\mathcal{H}^2$-matrix methods: *S. Börm, W. Hackbusch, B. Khoromskijet, S. Sauter.*

2004 Inversion of "FMM structure": *S. Chandrasekharan, T. Pals.*

2004 Proofs of compressibility: *M. Bebendorf, S. Börm, W. Hackbusch, ...*

2006 Accelerated nested diss. via $\mathcal{H}$-mats: *L. Grasedyck, R. Kriemann, S. LeBorne* [2007] *S. Chandrasekharan, M. Gu, X.S. Li, J. Xia.* [2010], *P. Schmitz and L. Ying.*

2010 construction of $\mathbf{A}^{-1}$ via randomized sampling: *L. Lin, J. Lu, L. Ying.*

Current status — problems with non-oscillatory kernels (Laplace, elasticity, Stokes, *etc*).

Problems on 1D domains:

- *Integral equations on the line:* Done. $O(N)$ with very small constants.
- *Boundary Integral Equations in $\mathbb{R}^2$:* Done. $O(N)$ with small constants.
- *BIEs on axisymmetric surfaces in $\mathbb{R}^3$:* Done. $O(N)$ with small constants.

Problems on 2D domains:

- *"FEM" matrices for elliptic PDEs in the plane:* $O(N)$ algorithms exist. Work remains.
- *Volume Int. Eq. in the plane (e.g. low frequency Lippman-Schwinger):* $O(N\,(\log N)^p)$ algorithms exist. $O(N)$ and high accuracy methods are under development.
- *Boundary Integral Equations in $\mathbb{R}^3$:* $O(N\,(\log N)^p)$ algorithms exist. $O(N)$ and high accuracy methods are under development.

Problems on 3D domains:

- *"FEM" matrices for elliptic PDEs:* Very active area!
  (Grasedyck & LeBorne; Michielssen; Xia; Ying; . . .)
- *Volume Int. Eq.:* Can be done, but requires a lot of memory.

Current status — problems with oscillatory kernels (Helmholtz, time-harmonic Maxwell, etc.).

*Direct solvers are **extremely** desirable in this environment!*

Problems on 1D domains:

- *Integral equations on the line:* Done — $O(N)$ with small constants.
- *Boundary Integral Equations in $\mathbb{R}^2$:* ???
- *("Elongated" surfaces in $\mathbb{R}^2$ and $\mathbb{R}^3$:* Done — $O(N \log N)$.)

Problems on 2D domains:

- *"FEM" matrices for Helmholtz equation in the plane:* ???
  ($O(N^{1.5})$ inversion is possible.)
- *Volume Int. Eq. in the plane (e.g. high frequency Lippman-Schwinger):* ???
- *Boundary Integral Equations in $\mathbb{R}^3$:* ???

Problems on 3D domains:

- ???? ($O(N^2)$ inversion sometimes possible — memory requirement is a concern.)

Recent work by B. Engquist and L. Ying — very efficient pre-conditioners based on structured matrix calculations. "Semi-direct."

How do these algorithms actually work?

Let us consider the simplest case: fast inversion of an equation on a 1D domain.

Things are *still* very technical ... quite involved notation ...

What follows is a brief description of a method from an extreme birds-eye view.

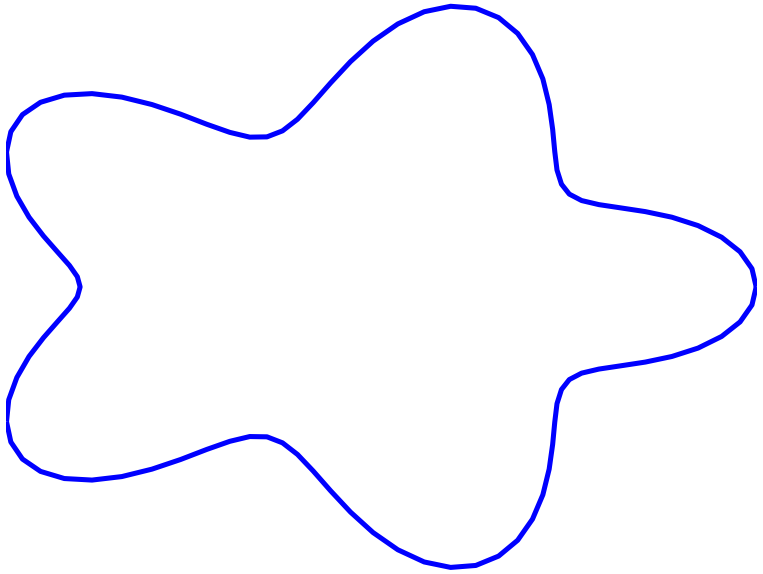
We start by describing some key properties of the matrices under consideration.

For concreteness, consider a 100×100 matrix $\mathbf{A}$ approximating the operator

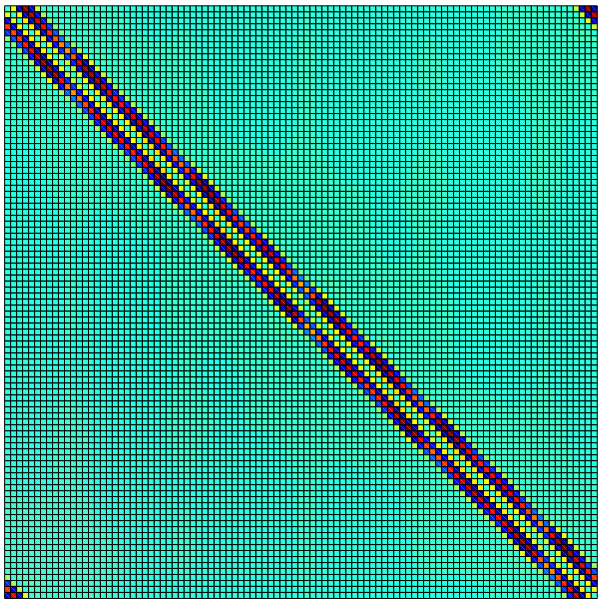
$$[\mathcal{S}_\Gamma\, u](\mathbf{x}) = u(\mathbf{x}) + \int_\Gamma \log|\mathbf{x} - \mathbf{y}|\, u(\mathbf{y})\, ds(\mathbf{y}).$$

The matrix $\mathbf{A}$ is characterized by:

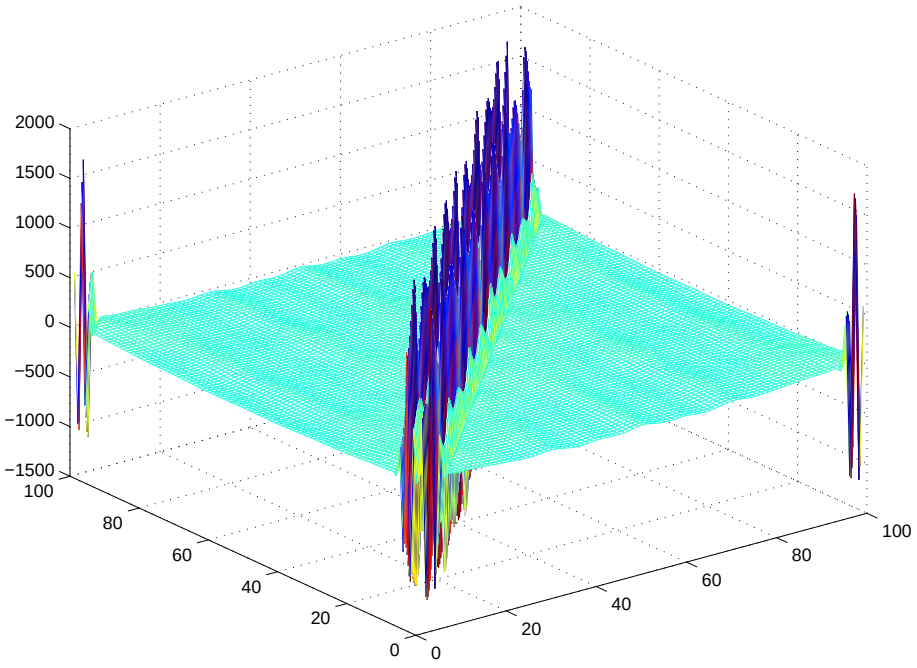
- Irregular behavior near the diagonal.
- Smooth entries away from the diagonal.



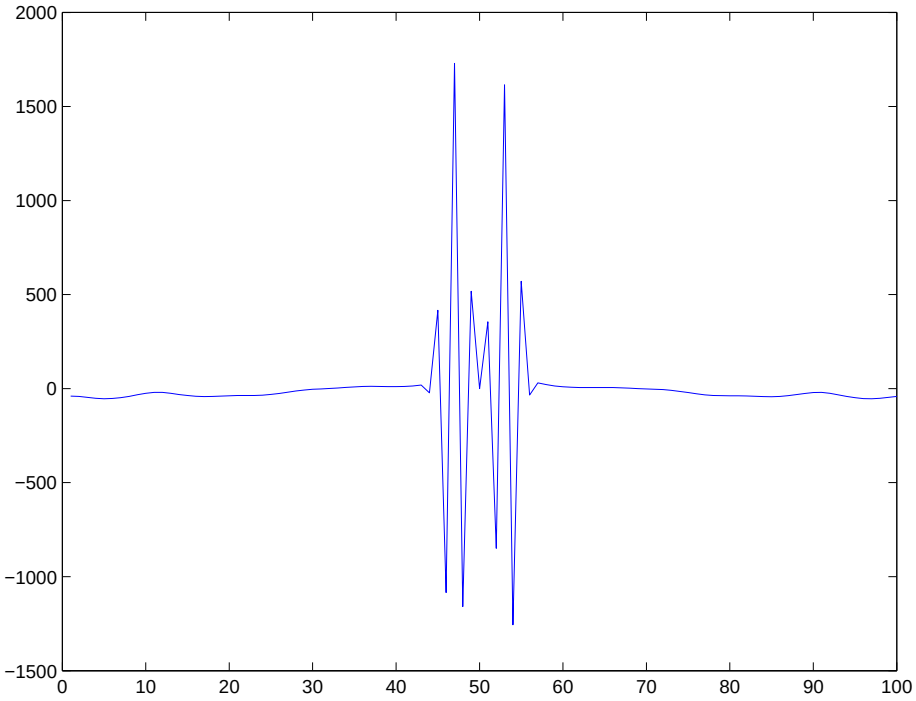
The contour Γ.



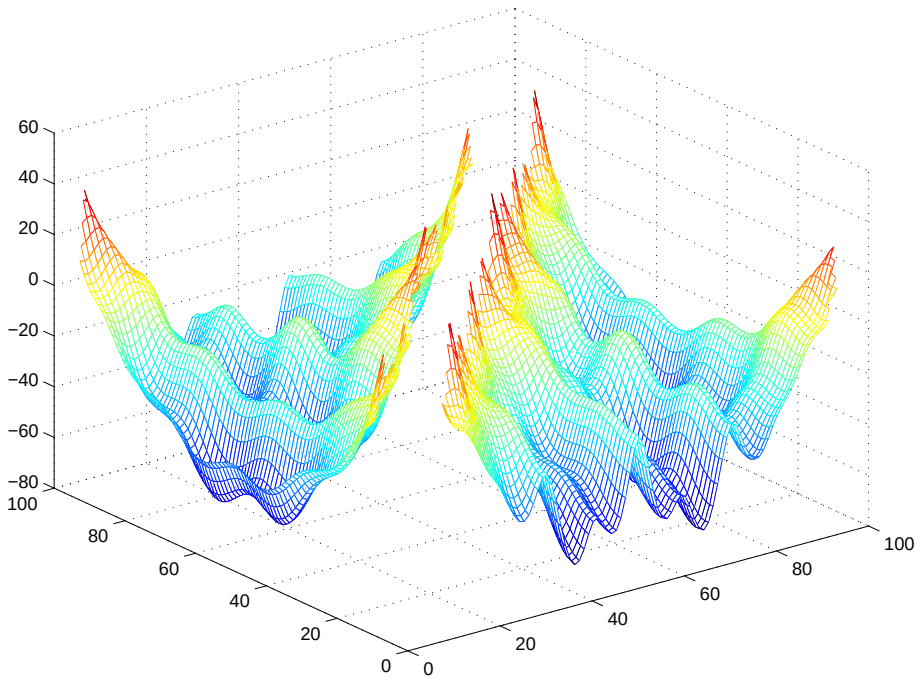
The matrix $\mathbf{A}$.



Plot of a_{ij} vs i and j

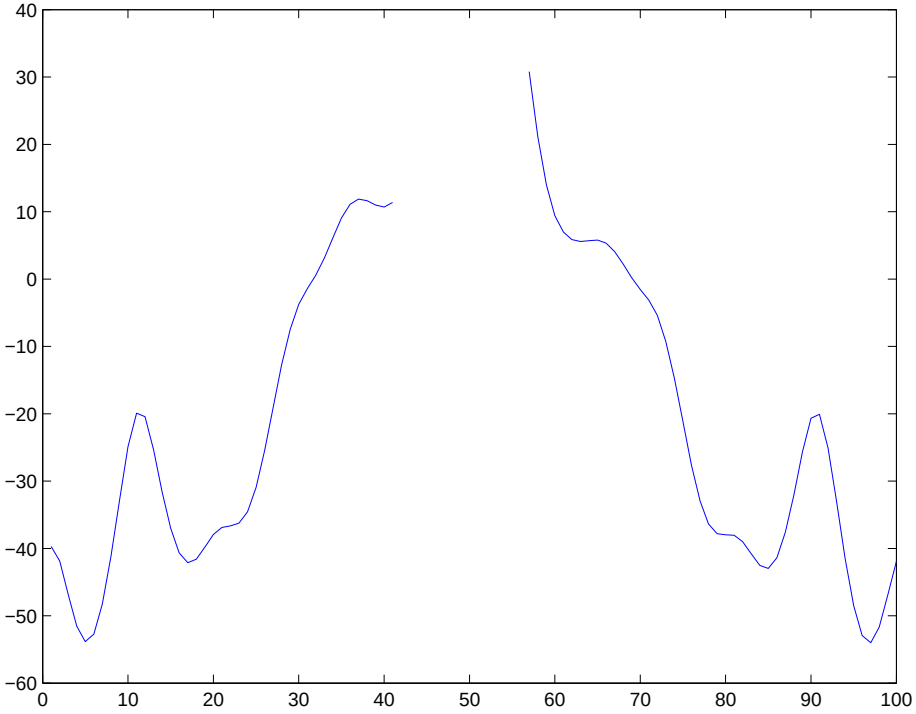


The 50^{th} row of **A**



Plot of a_{ij} vs i and j

(without the diagonal entries)

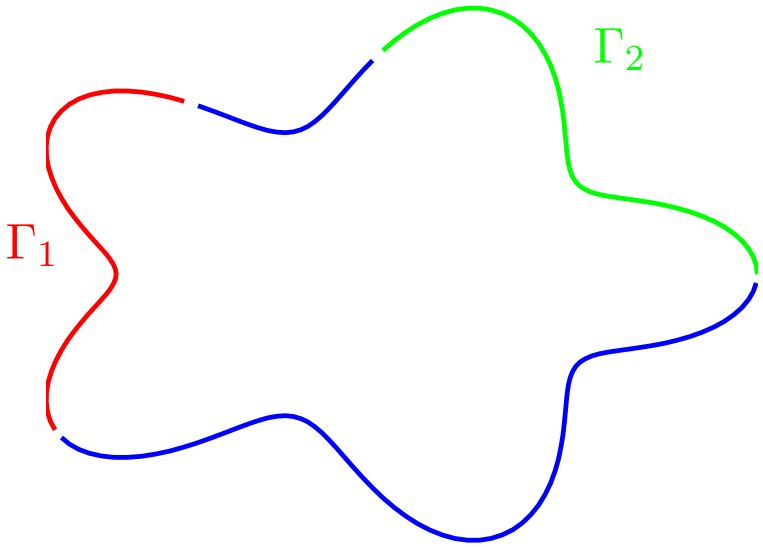


The 50^{th} row of $\mathbf{A}$

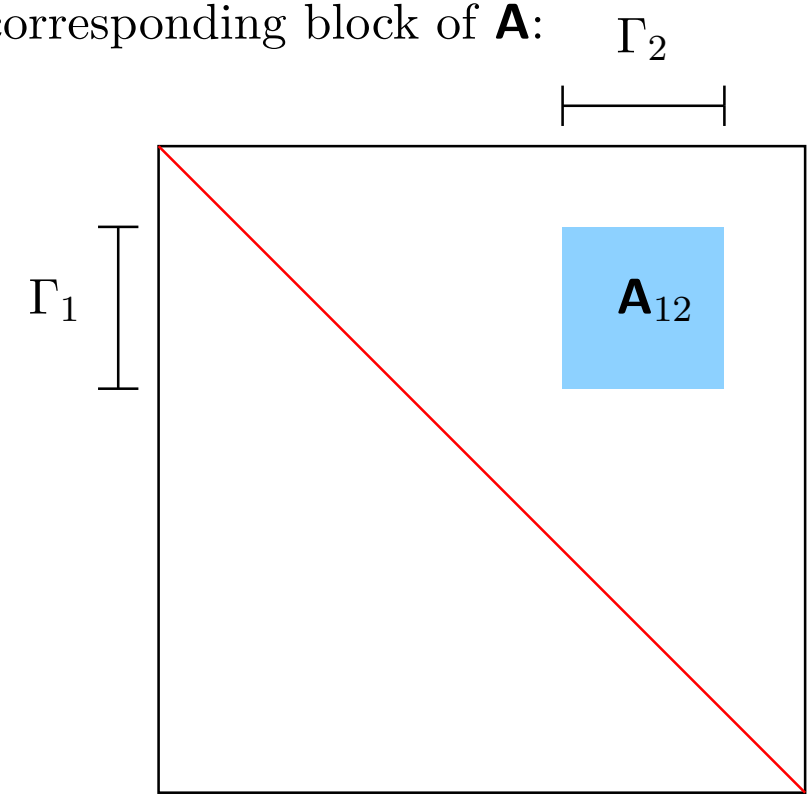
(without the diagonal entries)

Key observation: Off-diagonal blocks of **A** have low rank.

Consider two patches Γ_1 and Γ_2 and the corresponding block of **A**:



The contour Γ

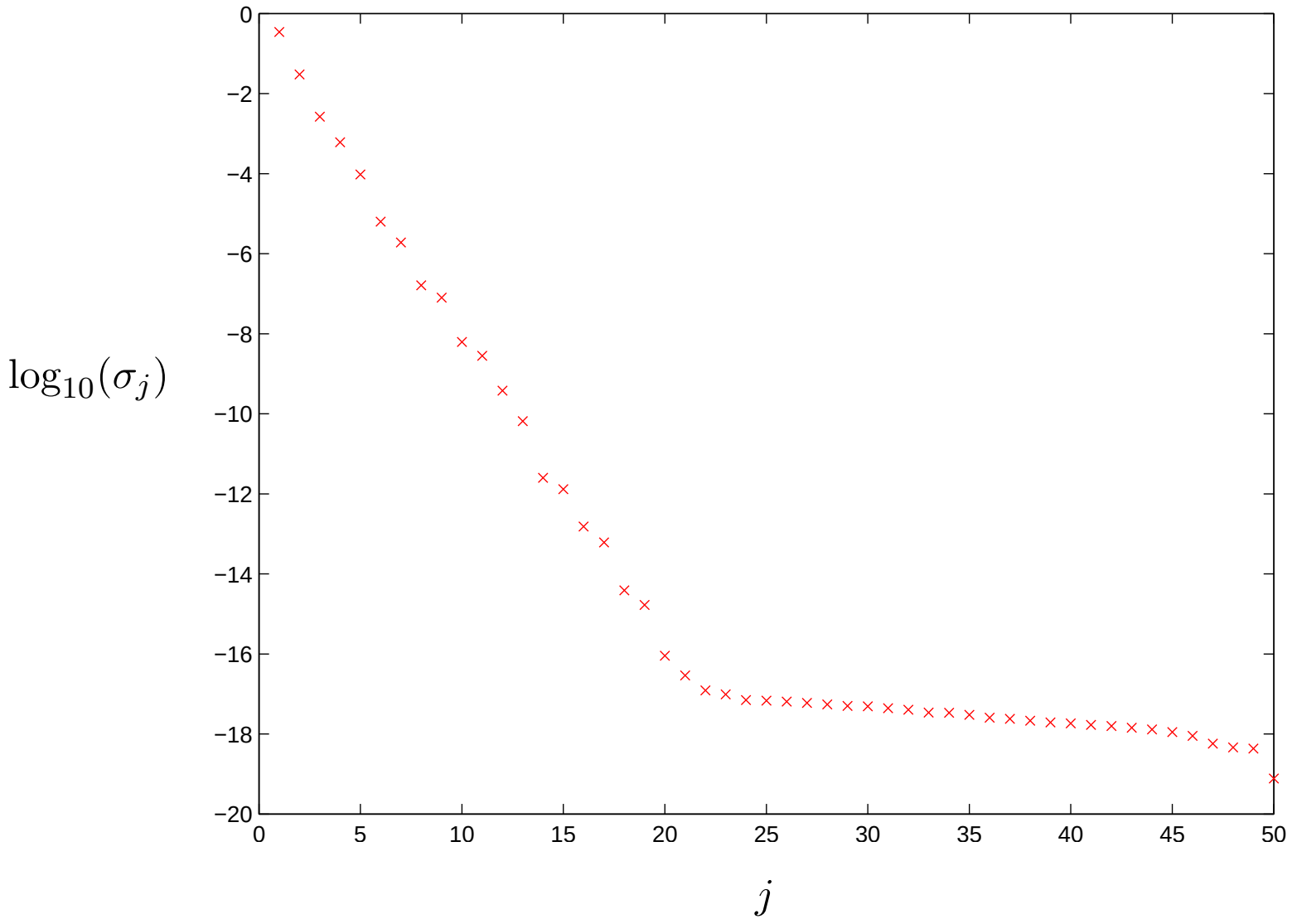


The matrix **A**

The block $\mathbf{A}_{12}$ is a discretization of the integral operator

$$[\mathcal{S}_{\Gamma_1 \leftarrow \Gamma_2}\, u](\mathbf{x}) = u(\mathbf{x}) + \int_{\Gamma_2} \log|\mathbf{x}-\mathbf{y}|\, u(\mathbf{y})\, ds(\mathbf{y}), \qquad \mathbf{x} \in \Gamma_1.$$

Singular values of $\mathbf{A}_{12}$ (now for a 200×200 matrix $\mathbf{A}$):



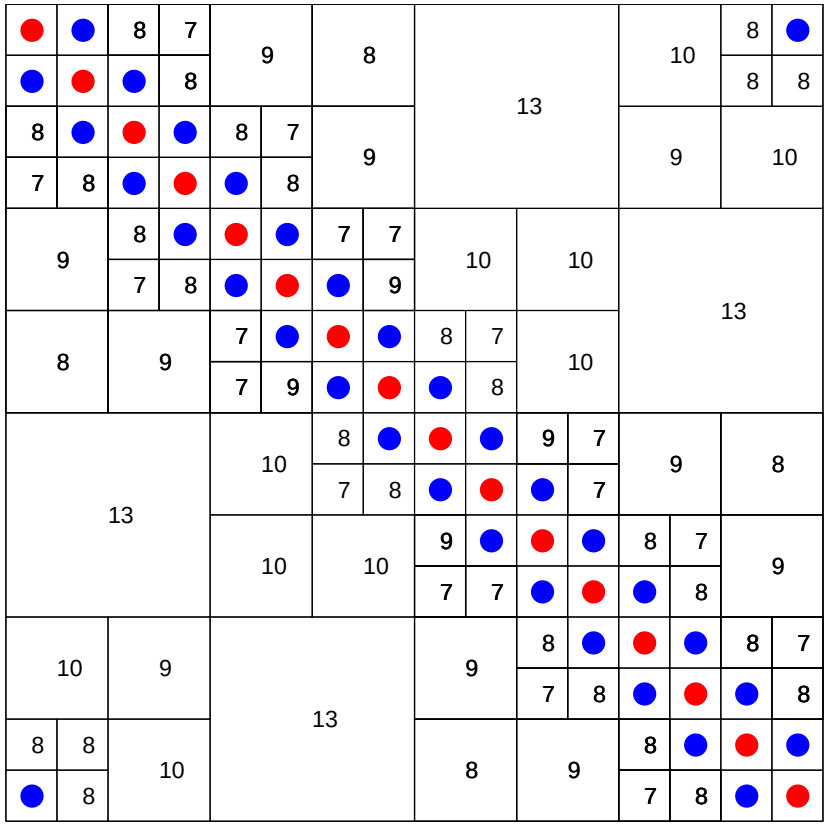
What we see is an artifact of the *smoothing effect* of coercive elliptic differential equations; it can be interpreted as a *loss of information.*

This effect has many well known physical consequences:

- The intractability of solving the heat equation backwards.
- The *St Venant principle* in mechanics.
- The inaccuracy of imaging at sub-wavelength scales.

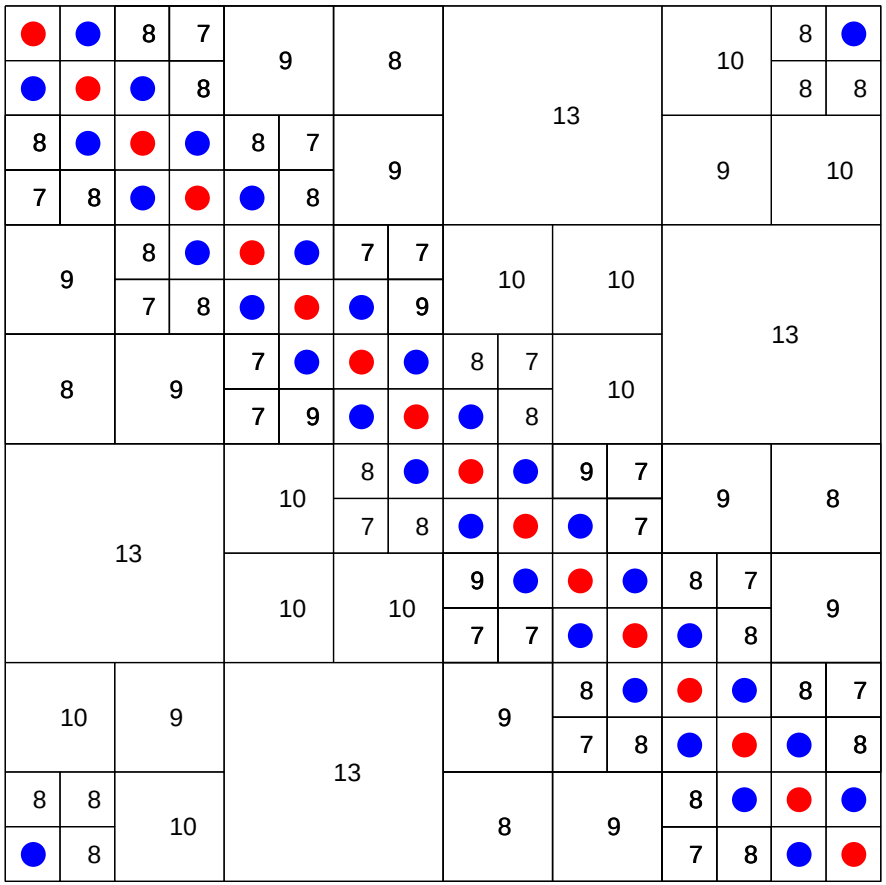
Such phenomena should be viewed in contrast to high-frequency scattering problems — extreme accuracy of optics *etc.*

Now that we know that off-diagonal blocks of **A** have low rank, all we need to do is to tessellate the matrix into as few such blocks are possible. A standard tessellation is:



The numbers shown are ranks to precision $\varepsilon = 10^{-10}$ *(*$N_{\text{tot}} = 800$*).*
Blocks with red and blue dots are stored as dense matrices.

Note how all blocks are well-separated from the diagonal. This is characteristic of both the Fast Multipole Method, and $\mathcal{H}$-matrix methods. (Our tessellation is slightly non-standard since it's based on a tree on *parameter space* rather than *physical space*.)



Storing the matrix:

$O(N \log N)$ is simple.
($\mathcal{H}$-matrix, Barnes-Hut)

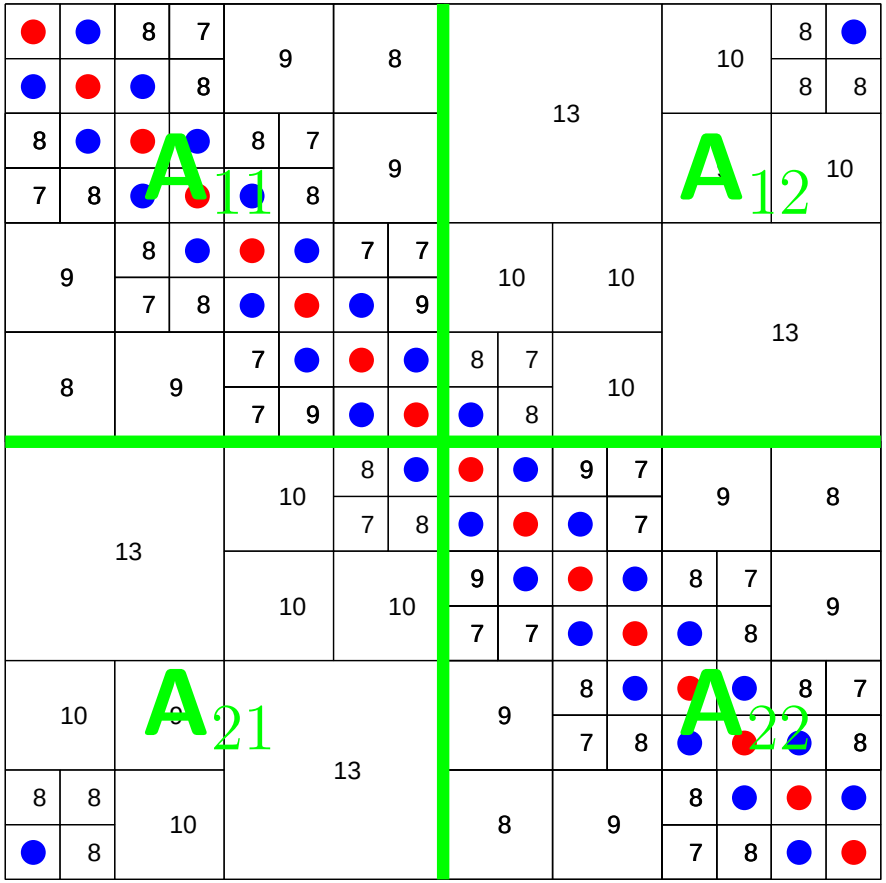
$O(N)$ is not that hard.
($\mathcal{H}^2$-matrix, FMM)

Matrix-vector multiply:

$O(N \log N)$ is simple.
($\mathcal{H}$-matrix, Barnes-Hut)

$O(N)$ is not that hard.
($\mathcal{H}^2$-matrix, FMM)

Matrix inversion: A bit more complicated.



Storing the matrix:

$O(N \log N)$ is simple.
($\mathcal{H}$-matrix, Barnes-Hut)

$O(N)$ is not that hard.
($\mathcal{H}^2$-matrix, FMM)

Matrix-vector multiply:

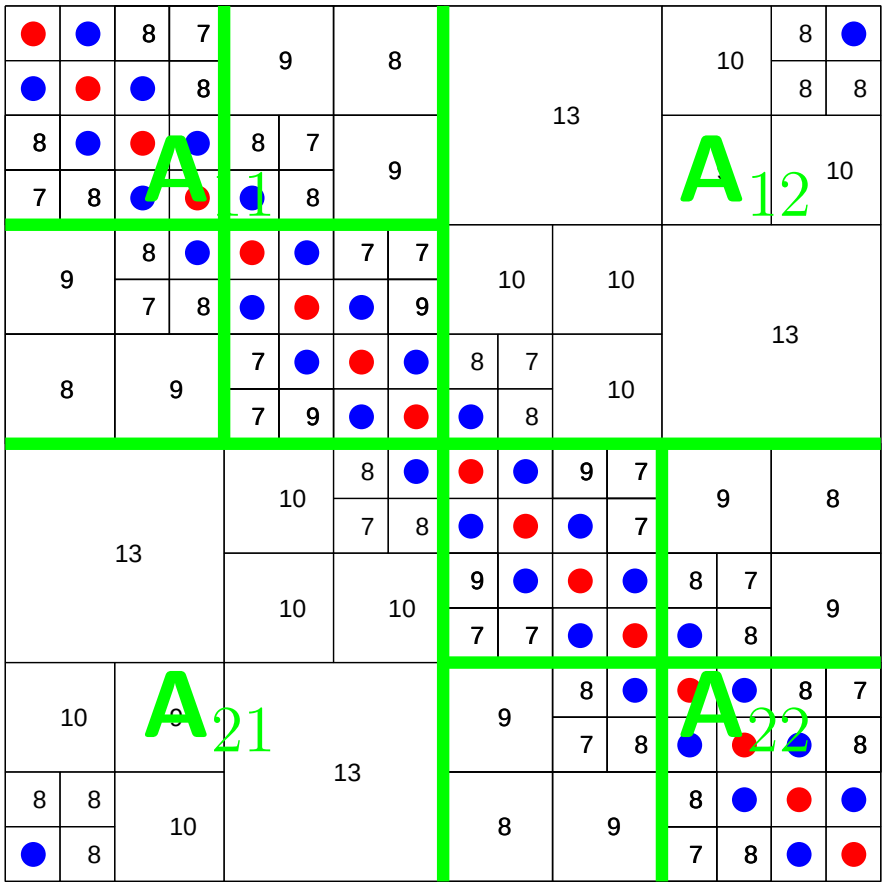
$O(N \log N)$ is simple.
($\mathcal{H}$-matrix, Barnes-Hut)

$O(N)$ is not that hard.
($\mathcal{H}^2$-matrix, FMM)

Matrix inversion: A bit more complicated.

With $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$. we have $\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{B}_{11}^{-1} & -\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \end{bmatrix}$,

where $\mathbf{B}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12}\,\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}$.



Storing the matrix:

$O(N \log N)$ is simple.
($\mathcal{H}$-matrix, Barnes-Hut)

$O(N)$ is not that hard.
($\mathcal{H}^2$-matrix, FMM)

Matrix-vector multiply:

$O(N \log N)$ is simple.
($\mathcal{H}$-matrix, Barnes-Hut)

$O(N)$ is not that hard.
($\mathcal{H}^2$-matrix, FMM)

Matrix inversion: A bit more complicated.

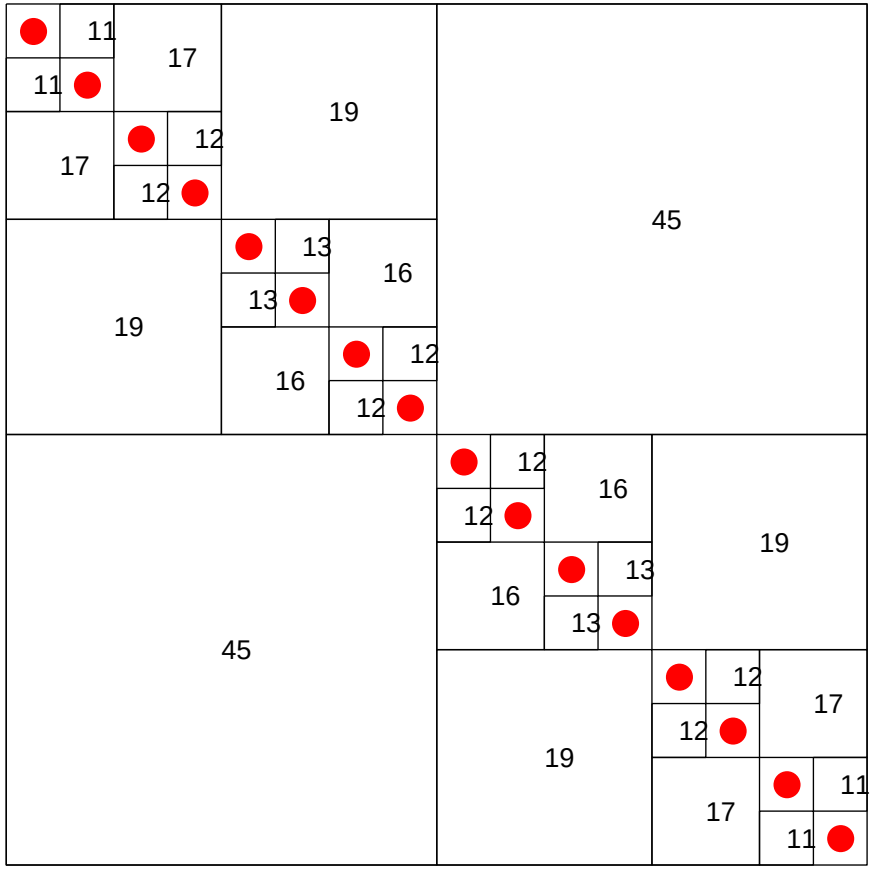
With $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$. we have $\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{B}_{11}^{-1} & -\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \end{bmatrix}$,

where $\mathbf{B}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12}\,\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}$.

Recurse for $O(N(\log N)^2)$ (?) complexity.

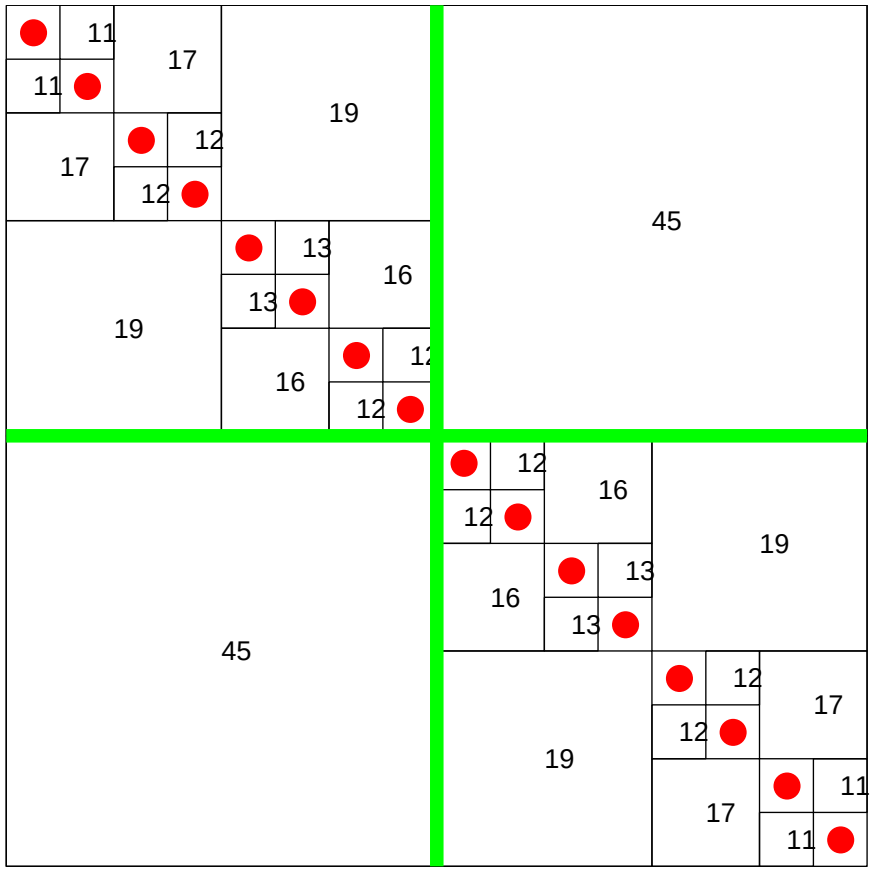
We are in luck! The 1D case has a remarkable property:

Even off-diagonal blocks that touch the diagonal have low rank.



We are in luck! The 1D case has a remarkable property:

Even off-diagonal blocks that touch the diagonal have low rank.



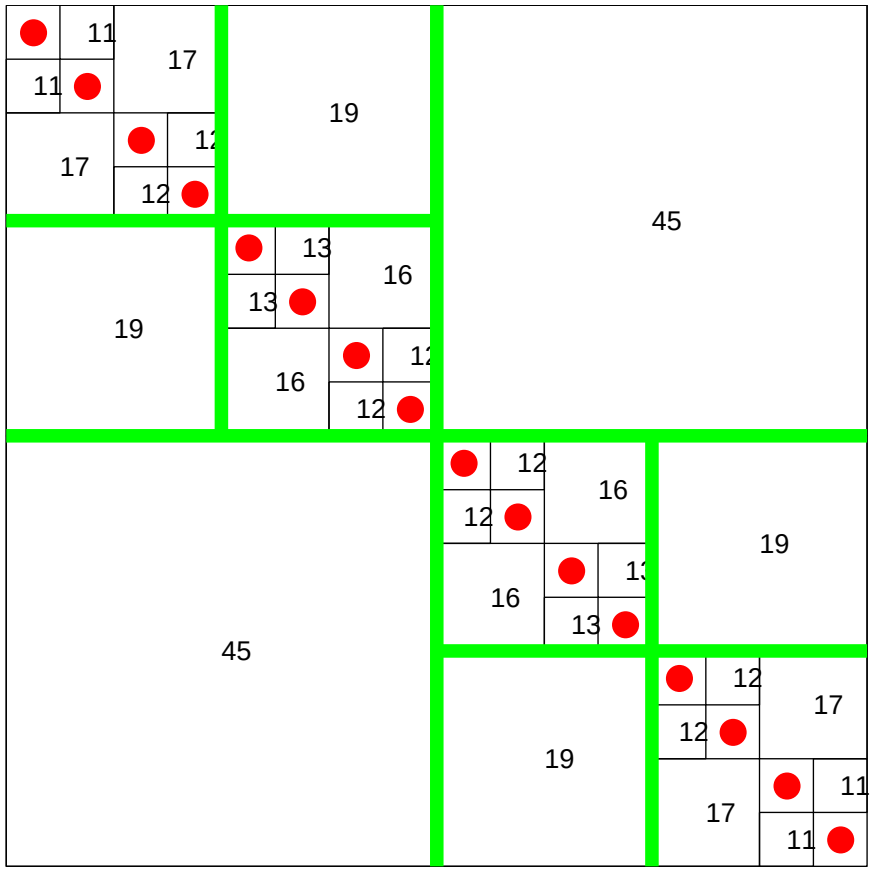
Matrix inversion: Simple!

With $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$. we have $\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{B}_{11}^{-1} & -\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \end{bmatrix}$,

where $\mathbf{B}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12}\,\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}$.

We are in luck! The 1D case has a remarkable property:

Even off-diagonal blocks that touch the diagonal have low rank.



Matrix inversion: Simple!

With $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$. we have $\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{B}_{11}^{-1} & -\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}\,\mathbf{B}_{11}^{-1}\,\mathbf{A}_{12}\,\mathbf{A}_{22}^{-1} \end{bmatrix}$,

where $\mathbf{B}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12}\,\mathbf{A}_{22}^{-1}\,\mathbf{A}_{21}$.

The "trick" of including blocks touching the diagonal has the advantages that it leads to much simpler algorithms, and less communication.

It has the slight disadvantage that the ranks increase somewhat.

It has a profound disadvantage in that standard (analytic) expansions of the kernel functions do not work. This problem has only been overcome in the last few years.

Direct solvers based on Hierarchically Semi-Separable matrices

Consider a linear system

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

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

$$\mathbf{A} = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14} \\ \mathbf{A}_{21} & \mathbf{D}_{22} & \mathbf{A}_{23} & \mathbf{A}_{24} \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{D}_{33} & \mathbf{A}_{34} \\ \mathbf{A}_{41} & \mathbf{A}_{42} & \mathbf{A}_{43} & \mathbf{D}_{44} \end{bmatrix}. \qquad \text{(Shown for } p = 4.\text{)}$$

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. (Say $k \approx n/2$.)

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).

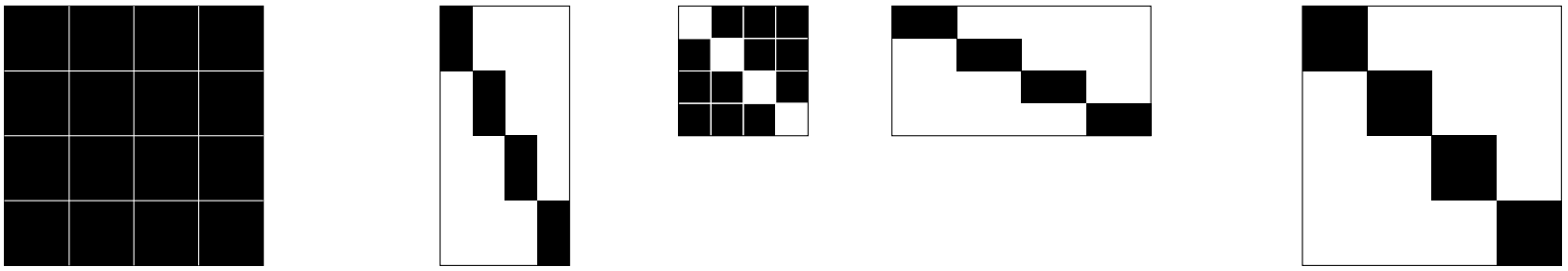
We get $\mathbf{A} = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \mathbf{V}_2^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{13} \mathbf{V}_3^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{14} \mathbf{V}_4^* \\ \mathbf{U}_2 \tilde{\mathbf{A}}_{21} \mathbf{V}_1^* & \mathbf{D}_{22} & \mathbf{U}_2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

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

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

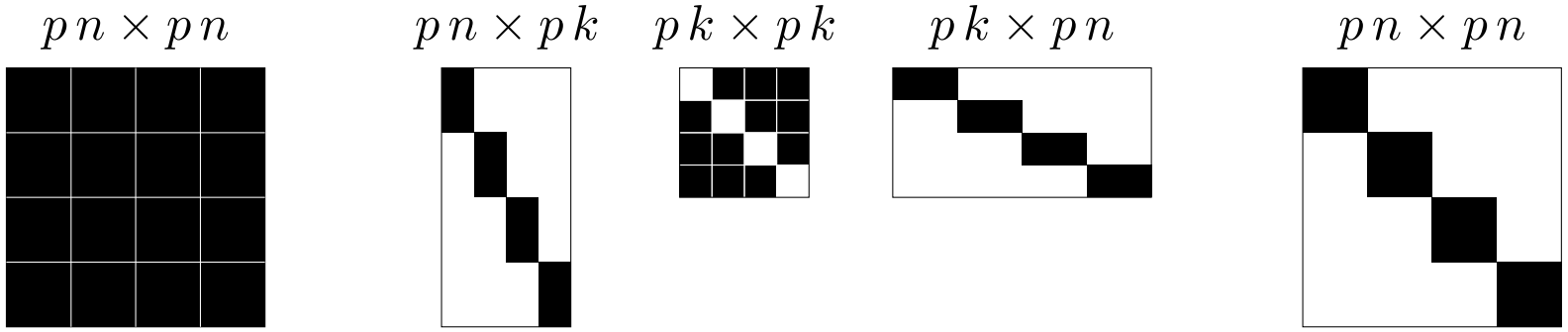
or

$$\underset{pn \times pn}{\mathbf{A}} = \underset{pn \times pk}{\mathbf{U}} \; \underset{pk \times pk}{\tilde{\mathbf{A}}} \; \underset{pk \times pn}{\mathbf{V}^*} + \underset{pn \times pn}{\mathbf{D}},$$



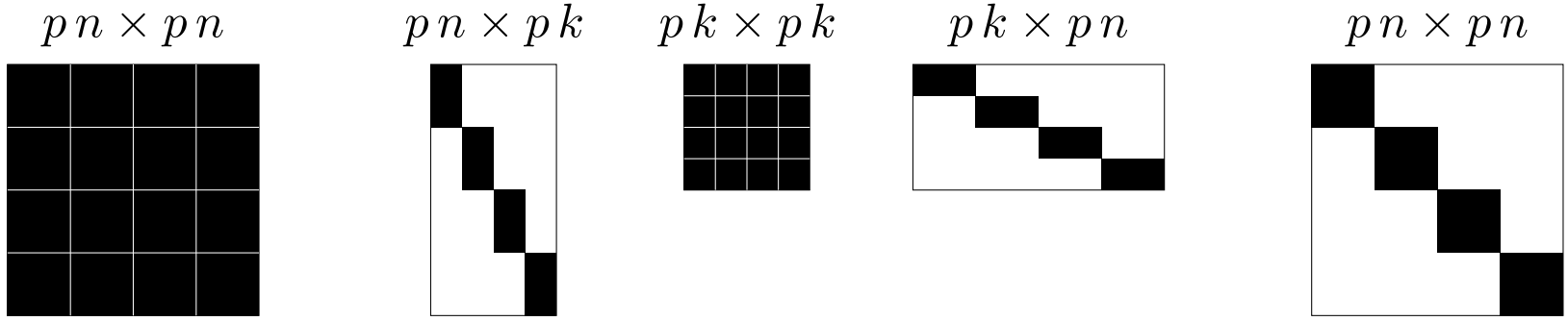
Lemma: [Variation of Woodbury] If an $N \times N$ matrix $\mathbf{A}$ admits the factorization

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



then

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



where (provided all intermediate matrices are invertible)

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

Note: All matrices set in blue are block diagonal.

The Woodbury formula replaces the task of inverting a $p\,n \times p\,n$ matrix by the task of inverting a $p\,k \times p\,k$ matrix.

The cost is reduced from $(p\,n)^3$ to $(p\,k)^3$.

We do not yet have a "fast" scheme ...

(Recall: $\mathbf{A}$ has $p \times p$ blocks, each of size $n \times n$ and of rank k.)

We must recurse!

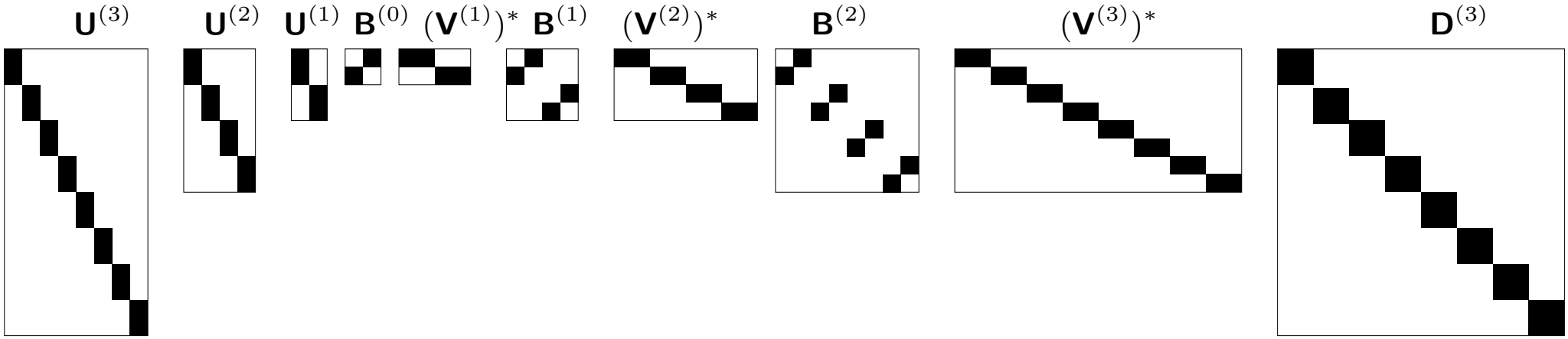
Using a telescoping factorization of $\mathbf{A}$ (a "hierarchically block-separable" representation):

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

we have a formula

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

Block structure of factorization:

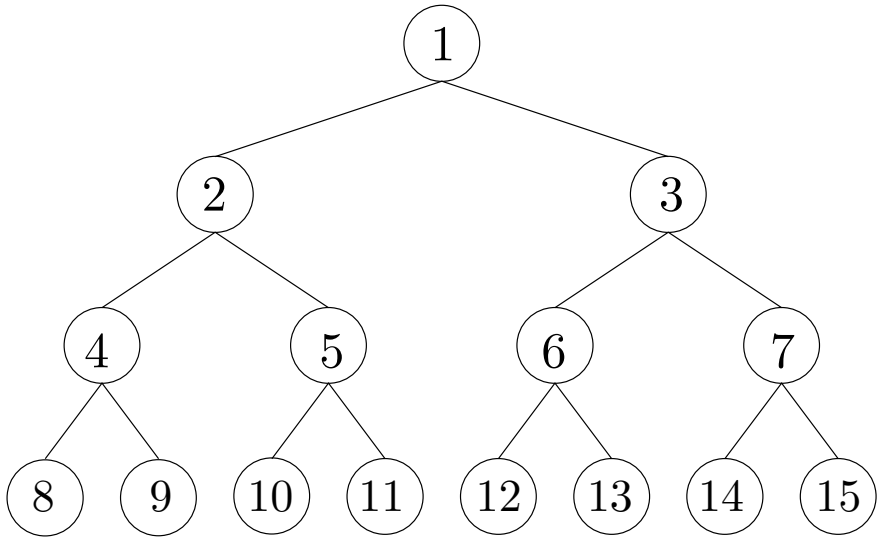


All matrices are now block diagonal except $\hat{\mathbf{D}}^{(0)}$*, which is small.*

Formal definition of an HSS matrix

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

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



Numbering of nodes in a fully populated binary tree with $L = 3$ levels.
The root is the original index vector $I = I_1 = [1, 2, \ldots, 400]$.

Formal definition of an HSS matrix

Suppose $\mathcal{T}$ is a binary tree.

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

For leaves σ and τ, set $\mathbf{A}_{\sigma,\tau} = \mathbf{A}(I_\sigma, I_\tau)$ and suppose that all off-diagonal blocks satisfy

$$\begin{array}{ccccccc} \mathbf{A}_{\sigma,\tau} & = & \mathbf{U}_\sigma & \tilde{\mathbf{A}}_{\sigma,\tau} & \mathbf{V}_\tau^* & & \sigma \neq \tau \\ n \times n & & n \times k & k \times k & k \times n & & \end{array}$$

For non-leaves σ and τ, let $\{\sigma_1, \sigma_2\}$ denote the children of σ, and let $\{\tau_1, \tau_2\}$ denote the children of τ. Set

$$\mathbf{A}_{\sigma,\tau} = \begin{bmatrix} \tilde{\mathbf{A}}_{\sigma_1,\tau_1} & \tilde{\mathbf{A}}_{\sigma_1,\tau_2} \\ \tilde{\mathbf{A}}_{\sigma_2,\tau_1} & \tilde{\mathbf{A}}_{\sigma_2,\tau_2} \end{bmatrix}$$

Then suppose that the off-diagonal blocks satisfy

$$\begin{array}{ccccccc} \mathbf{A}_{\sigma,\tau} & = & \mathbf{U}_\sigma & \tilde{\mathbf{A}}_{\sigma,\tau} & \mathbf{V}_\tau^* & & \sigma \neq \tau \\ 2k \times 2k & & 2k \times k & k \times k & k \times 2k & & \end{array}$$

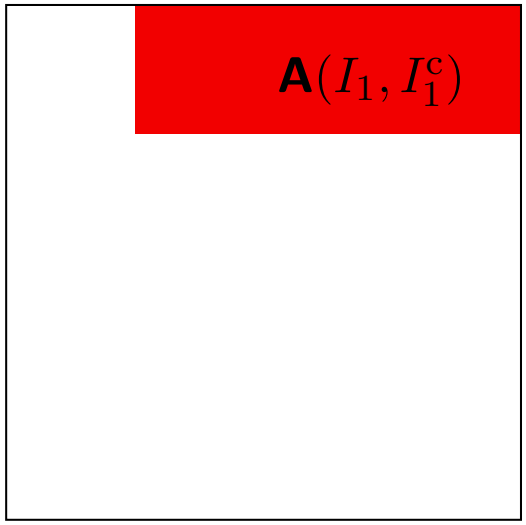
	Name:	Size:	Function:
For each leaf	$\mathbf{D}_\tau$	$n \times n$	The diagonal block $\mathbf{A}(I_\tau, I_\tau)$.
node τ:	$\mathbf{U}_\tau$	$n \times k$	Basis for the columns in the blocks in row τ.
	$\mathbf{V}_\tau$	$n \times k$	Basis for the rows in the blocks in column τ.
For each parent	$\mathbf{B}_\tau$	$2k \times 2k$	Interactions between the children of τ.
node τ:	$\mathbf{U}_\tau$	$2k \times k$	Basis for the columns in the (reduced) blocks in row τ.
	$\mathbf{V}_\tau$	$2k \times k$	Basis for the rows in the (reduced) blocks in column τ.

An HSS matrix $\mathbf{A}$ *associated with a tree* $\mathcal{T}$ *is fully specified if the factors listed above are provided.*

What is the role of the basis matrices $\mathbf{U}_\tau$ *and* $\mathbf{V}_\tau$*?*

Recall our toy example: $\mathbf{A} = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \mathbf{V}_2^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{13} \mathbf{V}_3^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{14} \mathbf{V}_4^* \\ \mathbf{U}_2 \tilde{\mathbf{A}}_{21} \mathbf{V}_1^* & \mathbf{D}_{22} & \mathbf{U}_2 \tilde{\mathbf{A}}_{23} \mathbf{V}_3^* & \mathbf{U}_2 \tilde{\mathbf{A}}_{24} \mathbf{V}_4^* \\ \mathbf{U}_3 \tilde{\mathbf{A}}_{31} \mathbf{V}_1^* & \mathbf{U}_3 \tilde{\mathbf{A}}_{32} \mathbf{V}_2^* & \mathbf{D}_{33} & \mathbf{U}_3 \tilde{\mathbf{A}}_{34} \mathbf{V}_4^* \\ \mathbf{U}_4 \tilde{\mathbf{A}}_{41} \mathbf{V}_1^* & \mathbf{U}_4 \tilde{\mathbf{A}}_{42} \mathbf{V}_2^* & \mathbf{U}_4 \tilde{\mathbf{A}}_{43} \mathbf{V}_3^* & \mathbf{D}_{44} \end{bmatrix}$.

We see that the columns of $\mathbf{U}_1$ must span the column space of the matrix $\mathbf{A}(I_1, I_1^c)$ where I_1 is the index vector for the first block and $I_1^c = I \backslash I_1$.



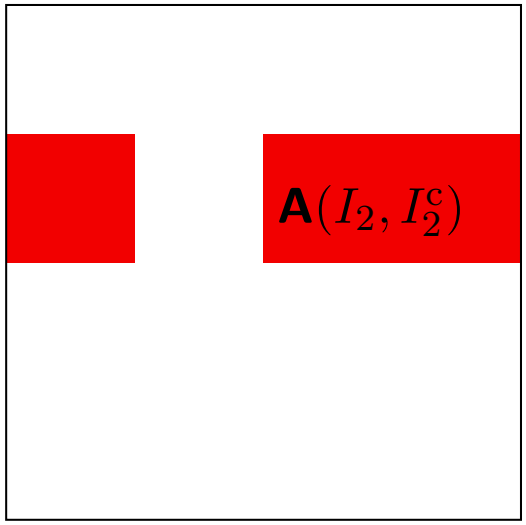
The matrix $\mathbf{A}$

What is the role of the basis matrices $\mathbf{U}_\tau$ *and* $\mathbf{V}_\tau$*?*

Recall our toy example: $\mathbf{A} =$

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

We see that the columns of $\mathbf{U}_2$ must span the column space of the matrix $\mathbf{A}(I_2, I_2^c)$ where I_2 is the index vector for the first block and $I_2^c = I \backslash I_2$.

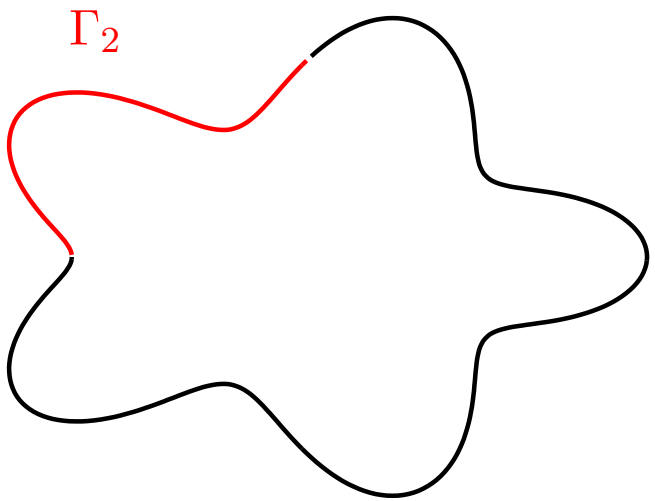


The matrix $\mathbf{A}$

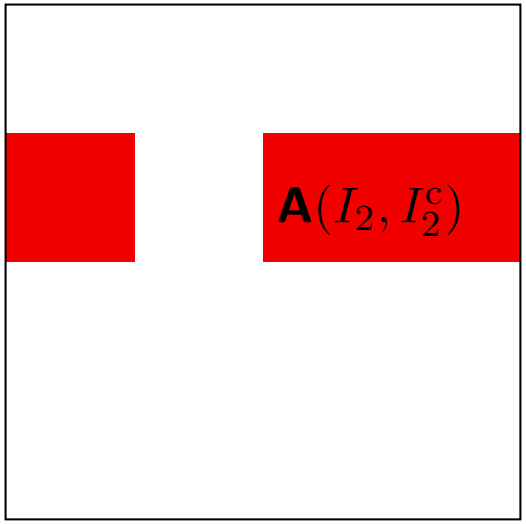
Let us consider a specific example.

Suppose that $\mathbf{A}$ is a discretization of the single layer operator

$$[\mathcal{S}_\Gamma\, u](\mathbf{x}) = u(\mathbf{x}) + \int_\Gamma \log|\mathbf{x}-\mathbf{y}|\, u(\mathbf{y})\, ds(\mathbf{y}).$$

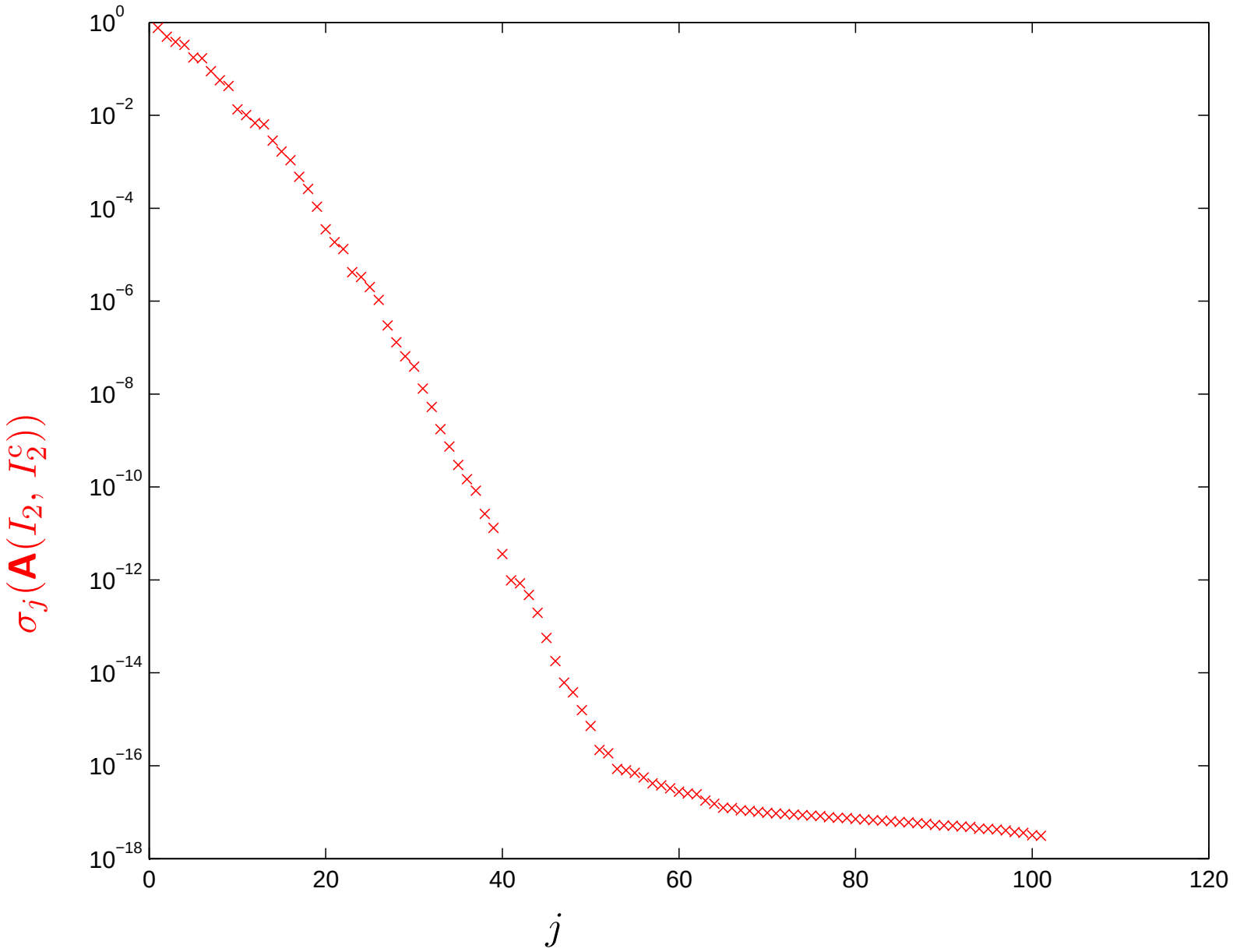


The contour Γ.



The matrix $\mathbf{A}$.

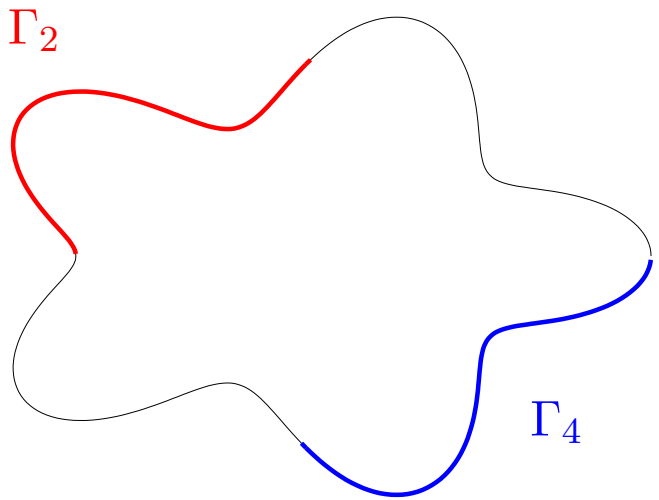
Singular values of $\mathbf{A}(I_2,\, I_2^{\mathrm{c}})$



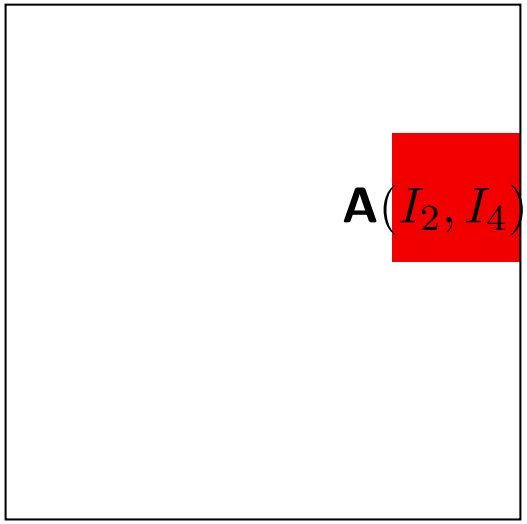
To precision 10^{-10}, the matrix $\mathbf{A}(I_2,\, I_2^{\mathrm{c}})$ has rank 36.

Remark: In an HSS representation, the ranks are typically higher than in an $\mathcal{H}$-matrix representation.

Specifically, the block $\mathbf{A}(I_2, I_2^c)$ would typically be considered "inadmissible."
Instead, in an $\mathcal{H}$-matrix representation, you would compress blocks such as $\mathbf{A}(I_2, I_4)$:

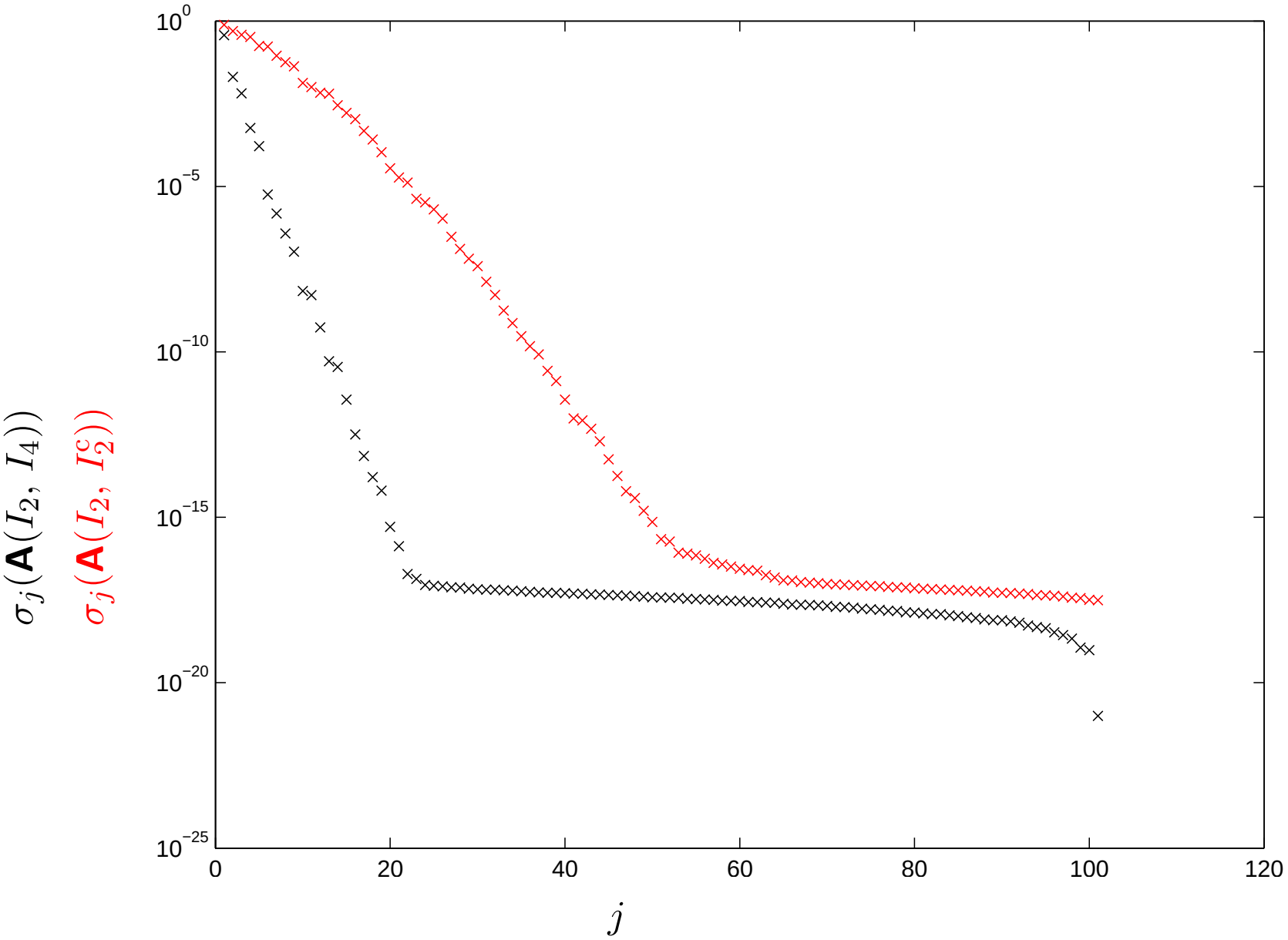


The contour Γ.



The matrix $\mathbf{A}$.

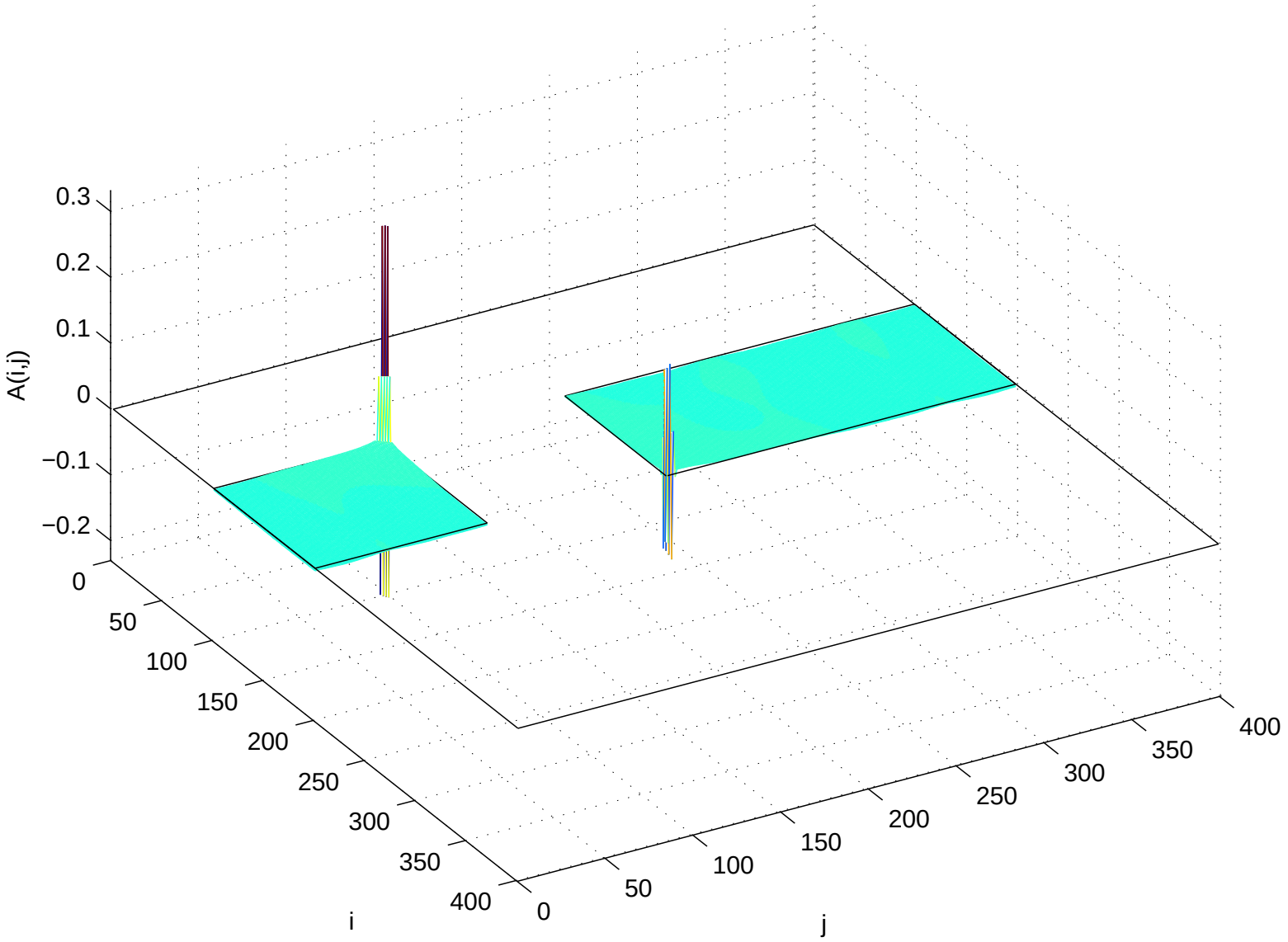
Singular values of $\mathbf{A}(I_2,\, I_2^{\mathrm{c}})$ *and* $\mathbf{A}(I_2,\, I_4)$*:*



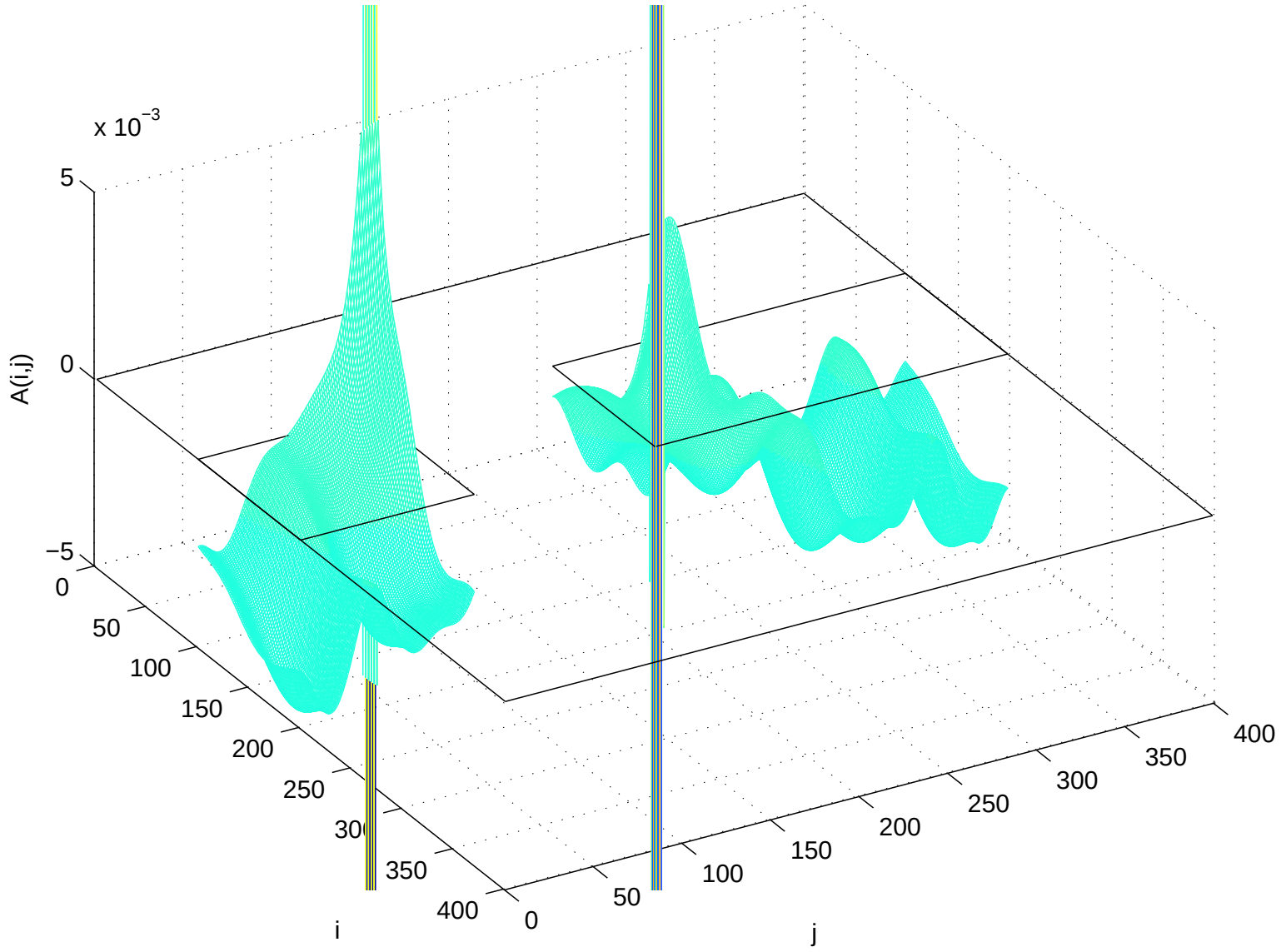
To precision 10^{-10}, the matrix $\mathbf{A}(I_2,\, I_2^{\mathrm{c}})$ has rank 36.

To precision 10^{-10}, the matrix $\mathbf{A}(I_2,\, I_4)$ has rank 12.

Plot of $\mathbf{A}(I_2, I_2^c)$

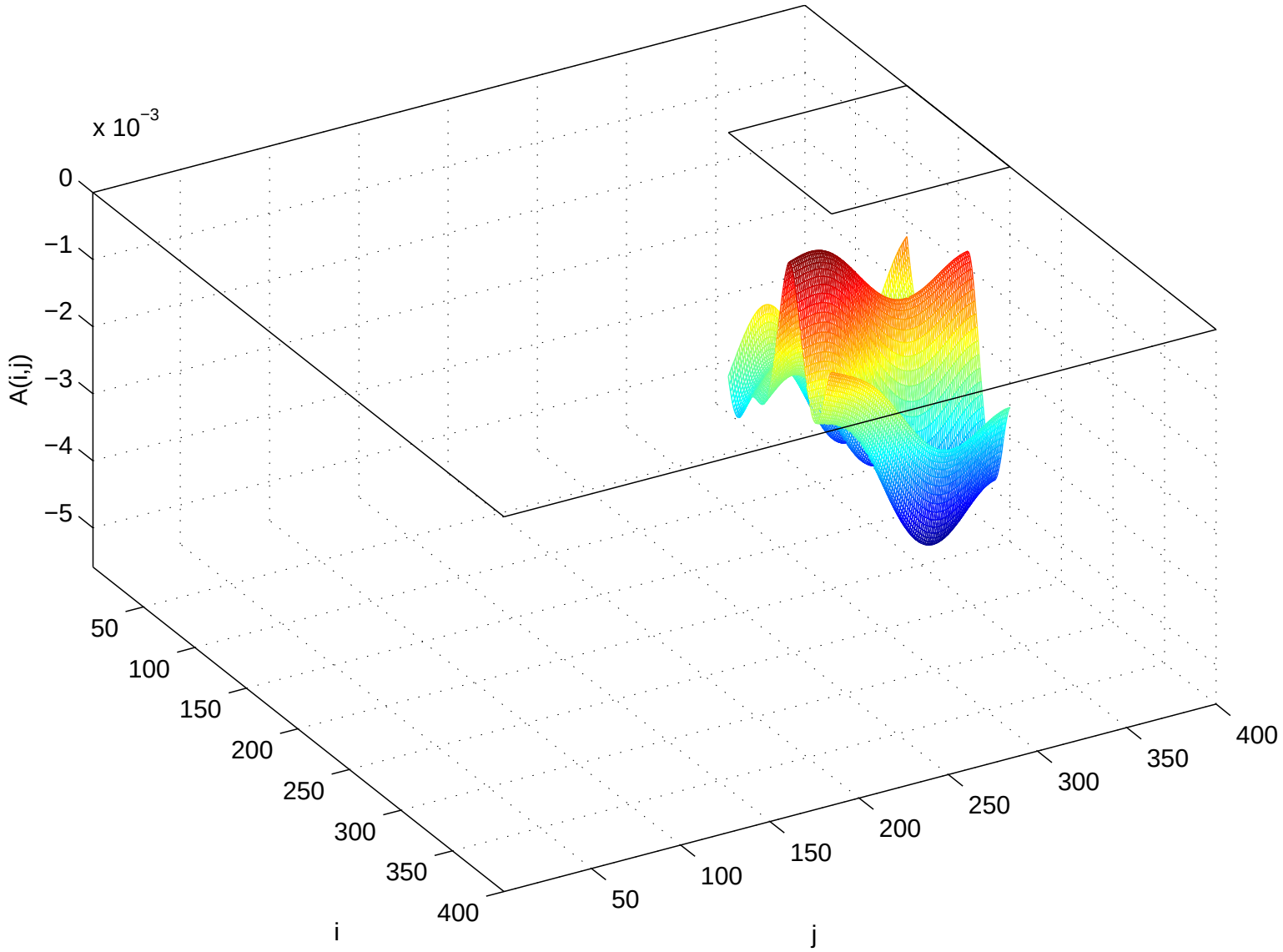


Plot of $\mathbf{A}(I_2,\, I_2^{\mathrm{c}})$



(Note: the z-axis has been rescaled)

Plot of $\mathbf{A}(I_2, I_4)$



Choice of basis matrices (our approach is non-standard):

Recall: The HSS structure relies on factorizations such as (for $k < n$)

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

For HSS matrix algebra to be numerically stable, it is critical that the basis matrices $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ be well-conditioned.

The gold-standard is to have $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ be orthonormal (i.e. $\sigma_j(\mathbf{U}_\tau) = \sigma_j(\mathbf{V}_\tau) = 1$ for $j = 1, 2, \ldots, k$), and this is commonly enforced.

We have decided to instead use *interpolatory decompositions* in which:

1. $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ each contain the $k \times k$ identity matrix as a submatrix.
2. $\mathbf{U}_\tau$ and $\mathbf{V}_\tau$ are "reasonably" well-conditioned.
3. $\tilde{\mathbf{A}}_{\sigma,\tau}$ is a submatrix of $\mathbf{A}$ for all σ, τ.

Our choice leads to some loss of accuracy, but vastly simplifies the task of computing compressed representations in the context of integral equations. (For instance, if the original $\mathbf{A}$ represents a Nyström discretization, then the HSS representation on each level is also a Nyström discretization, only with modified diagonal blocks, and on coarser discretizations.)

Numerical examples

All numerical examples were run on standard office desktops (most of them on an older 3.2 GHz Pentium IV with 2GB of RAM).

Most of the programs are written in Matlab (some in Fortran 77).

Recall that the reported CPU times have two components:

(1) Pre-computation (inversion, LU-factorization, constructing a Schur complement)

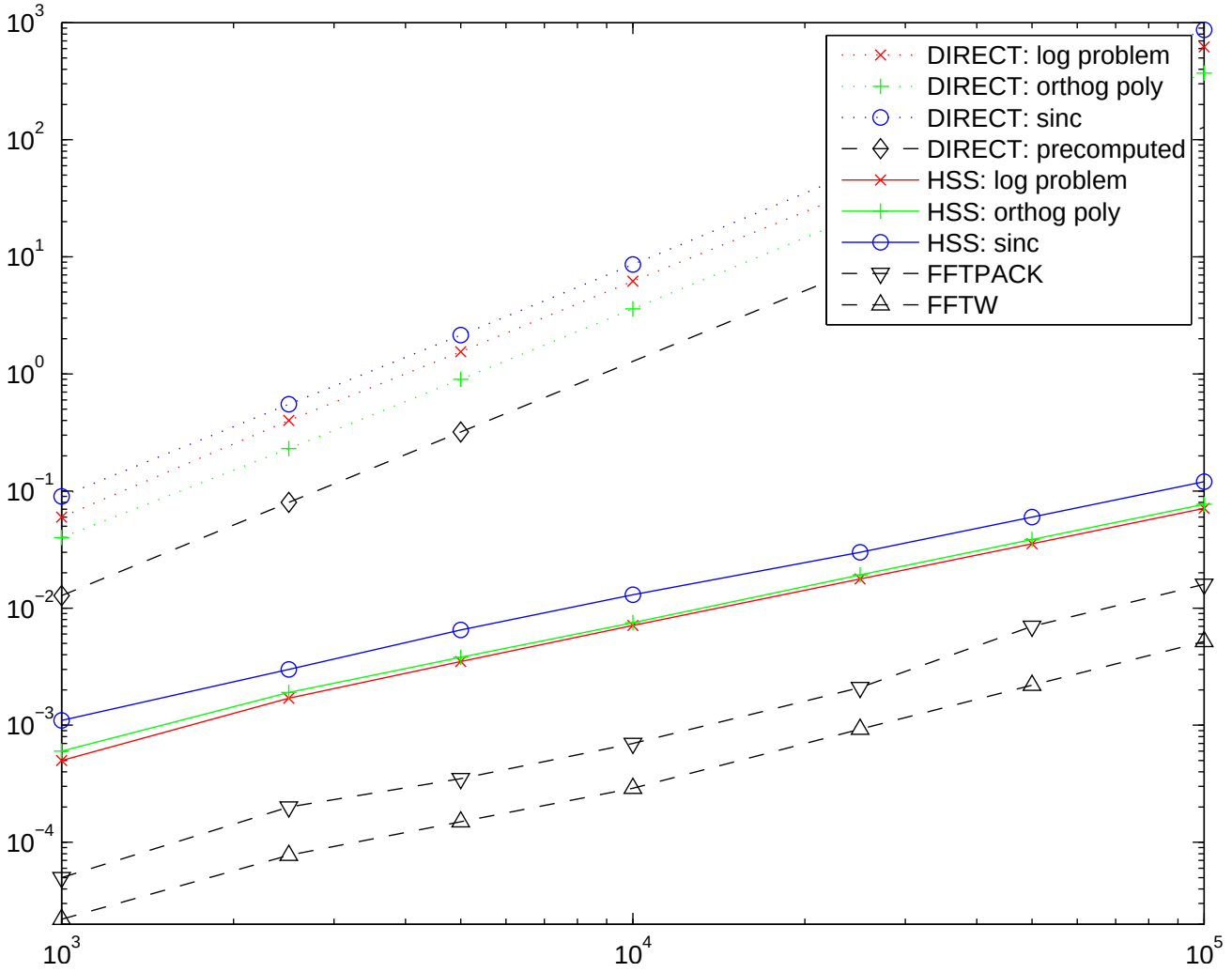
(2) Time for a single solve once pre-computation is completed

1D numerical examples — speed of HSS matvec (at accuracy 10^{-10})

Logarithmic kernel: $u_m = \sum_{\substack{n=1 \\ n \neq m}}^{N} \left(\log |x_m - x_n|\right) q_n$ with x_n drawn at random from $[0, 1]$.

Orthog poly: $u_m = \sum_{n=1}^{N} \frac{p_{k+1}(x_m)p_k(x_n) - p_k(x_m)p_{k+1}(x_n)}{x_m - x_n} q_n$ with $\{x_n\}_{n=1}^{N}$ Gaussian nodes.

Sinc kernel: $u_m = \sum_{n=1}^{N} \frac{\sin((x_m - x_n)\pi N/5)}{x_m - x_n} q_n$ with $(x_n)_{n=1}^{N}$ equispaced in $[-1, 1]$.



Note: Close to FFT speed! Break-even point with dense < 100!

1D numerical examples — BIEs in $\mathbb{R}^2$

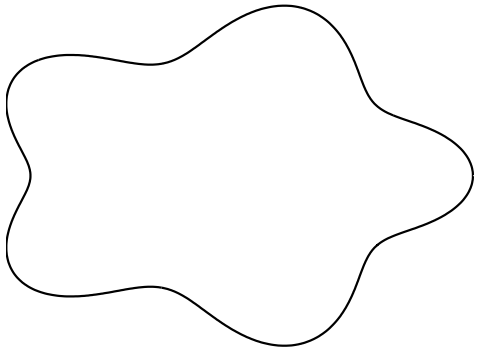
We invert a matrix approximating the operator

$$[A\,u](\mathbf{x}) = \frac{1}{2}\,u(\mathbf{x}) - \frac{1}{2\,\pi}\int_\Gamma D(\mathbf{x},\mathbf{y})\,u(\mathbf{y})\,ds(\mathbf{y}), \qquad \mathbf{x}\in\Gamma,$$

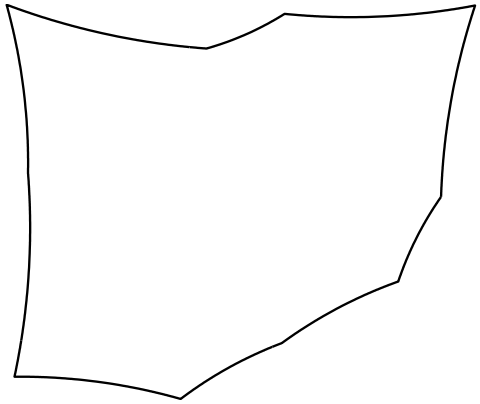
where D is the double layer kernel associated with Laplace's equation,

$$D(\mathbf{x},\mathbf{y}) = \frac{1}{2\pi}\,\frac{\mathbf{n}(\mathbf{y})\cdot(\mathbf{x}-\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^2},$$

and where Γ is either one of the countours:



Smooth star



Star with corners

(local refinements at corners)

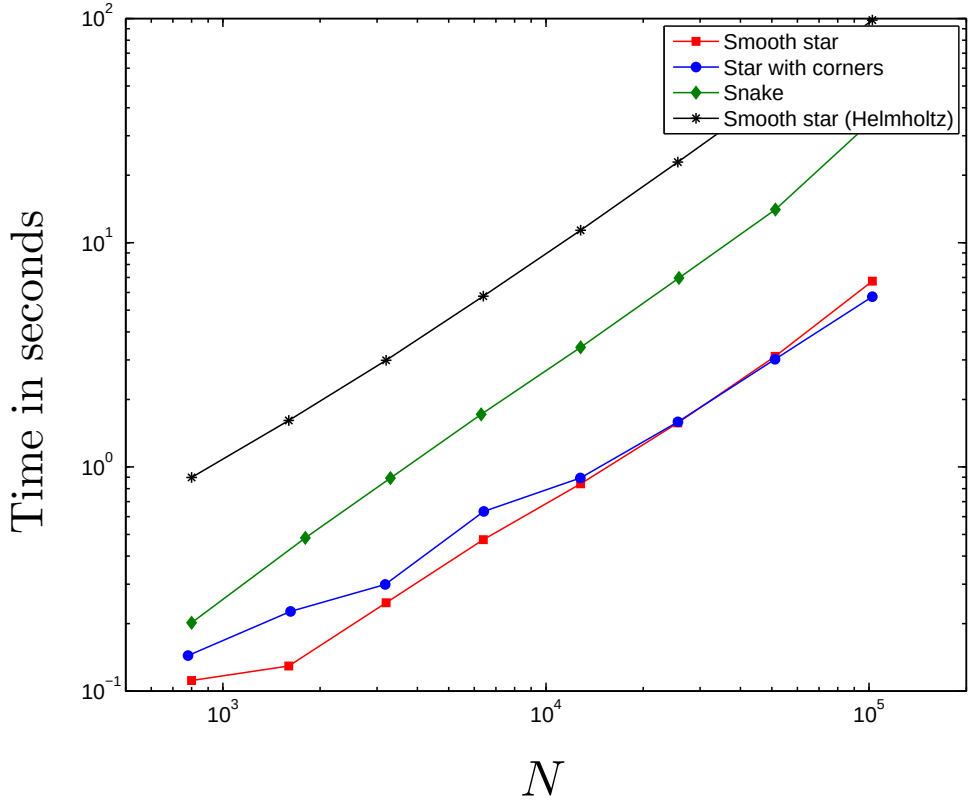
Snake

(# oscillations $\sim N$)

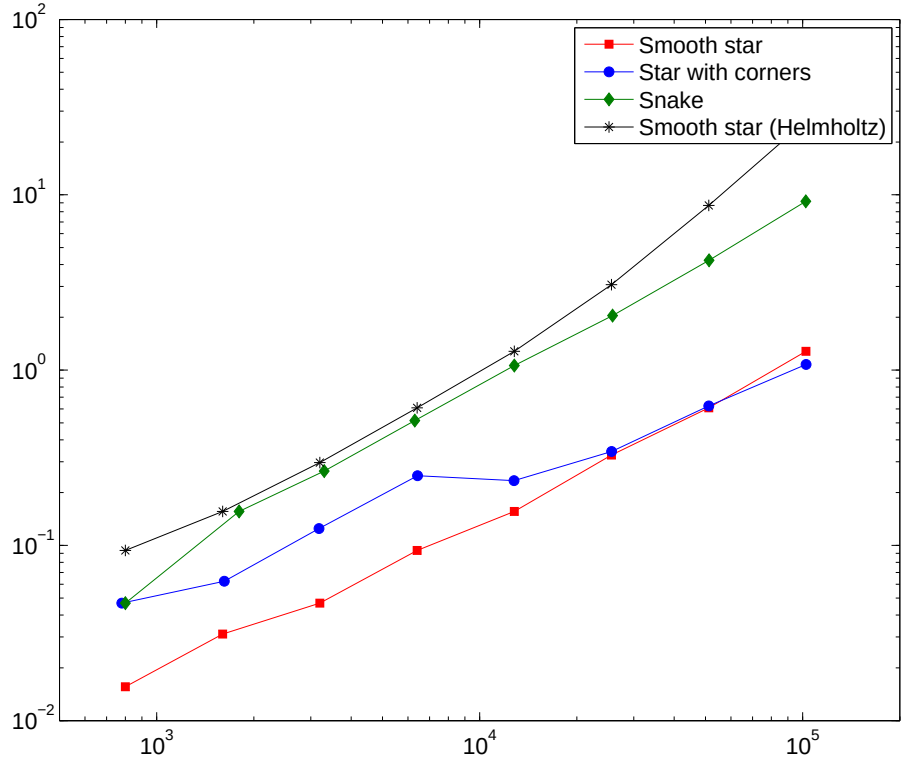
Examples from "A direct solver with O(N) complexity for integral equations on one-dimensional domains," A. Gillman, P. Young, P.G. Martinsson, 2011, Frontiers of Mathematics in China.

1D numerical examples — BIEs in $\mathbb{R}^2$

Compression



Inversion



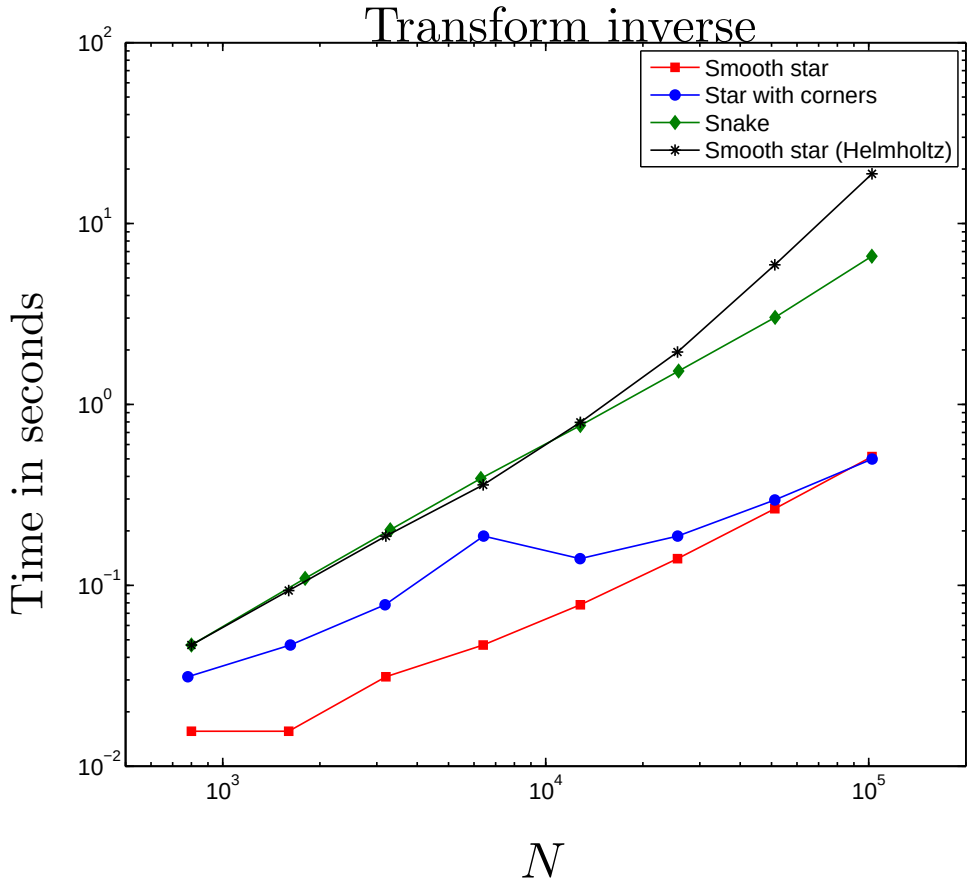
The graphs give the times required for:

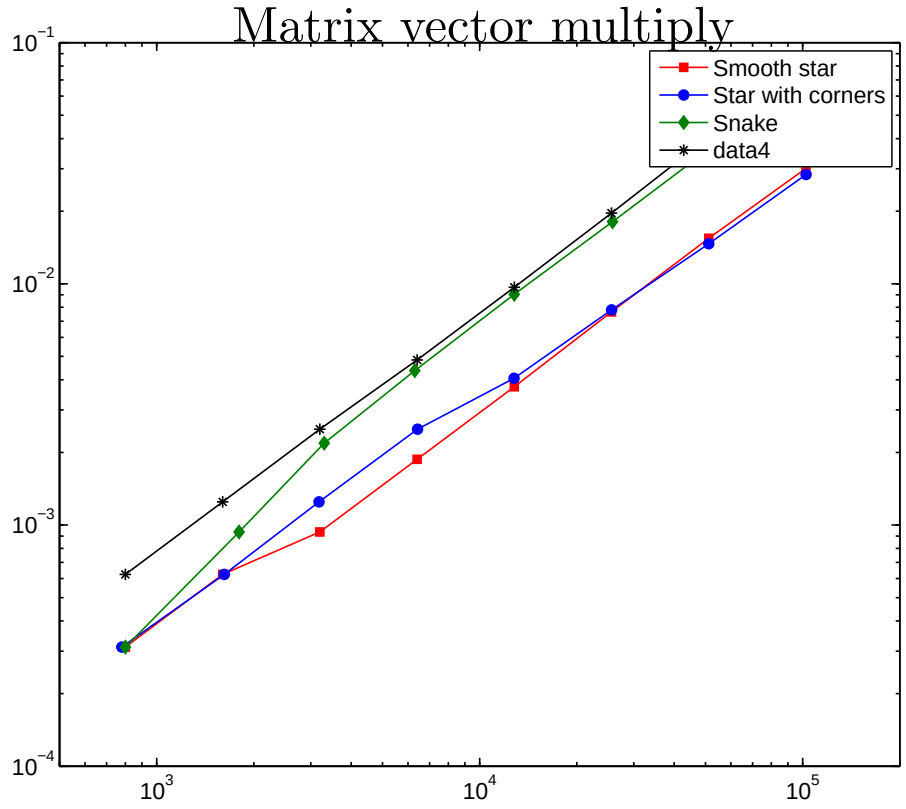
- Computing the HSS representation of the coefficient matrix.
- Inverting the HSS matrix.

Within each graph, the four lines correspond to the four examples considered:

□ Smooth star ○ Star with corners ◇ Snake ∗ Smooth star (Helmholtz)

1D numerical examples — BIEs in $\mathbb{R}^2$





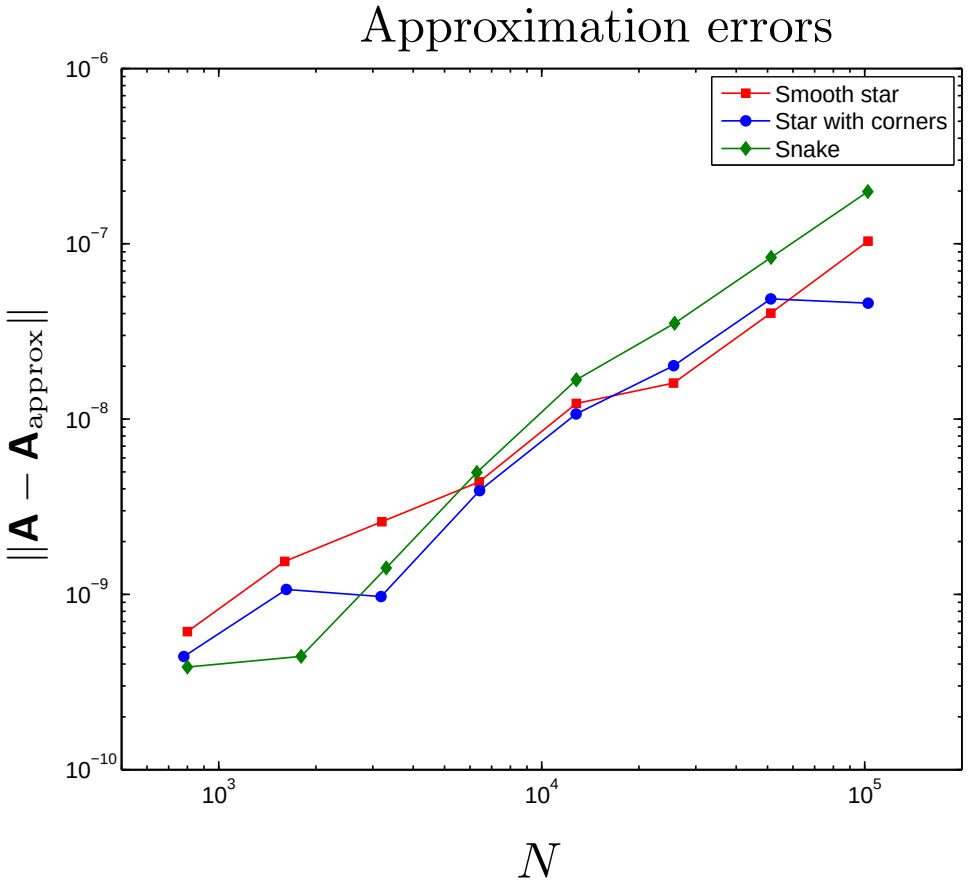
The graphs give the times required for:

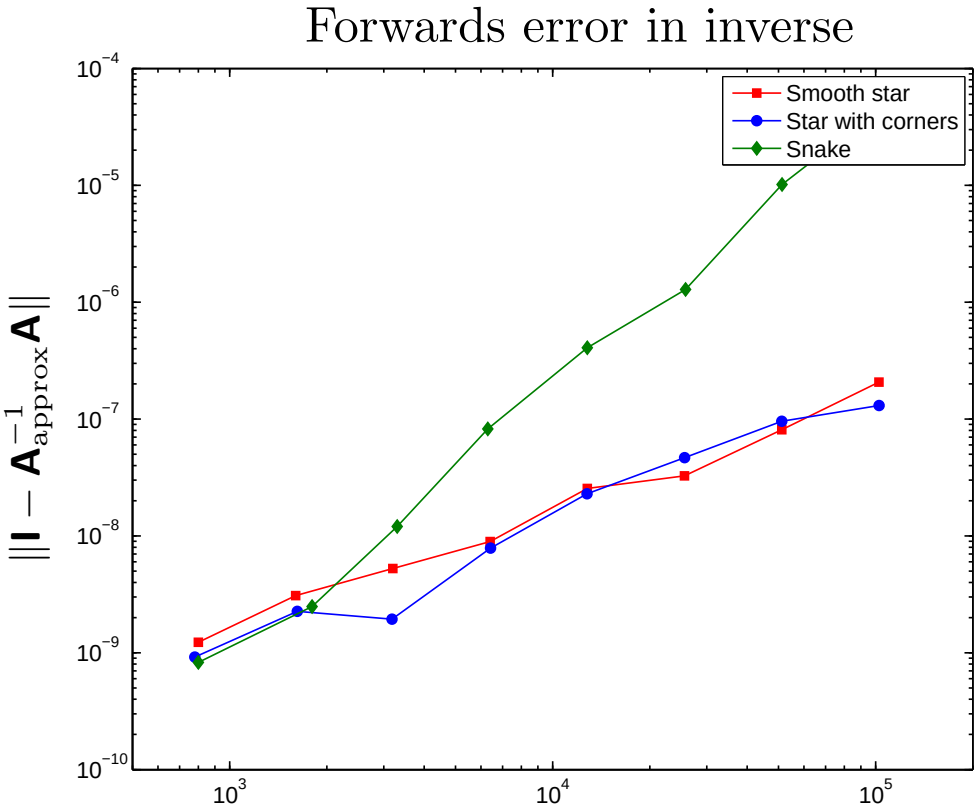
- Transforming the computed inverse to standard HSS format.
- Applying the inverse to a vector (i.e. solving a system).

Within each graph, the four lines correspond to the four examples considered:

□ Smooth star ○ Star with corners ◇ Snake ∗ Smooth star (Helmholtz)

1D numerical examples — BIEs in $\mathbb{R}^2$



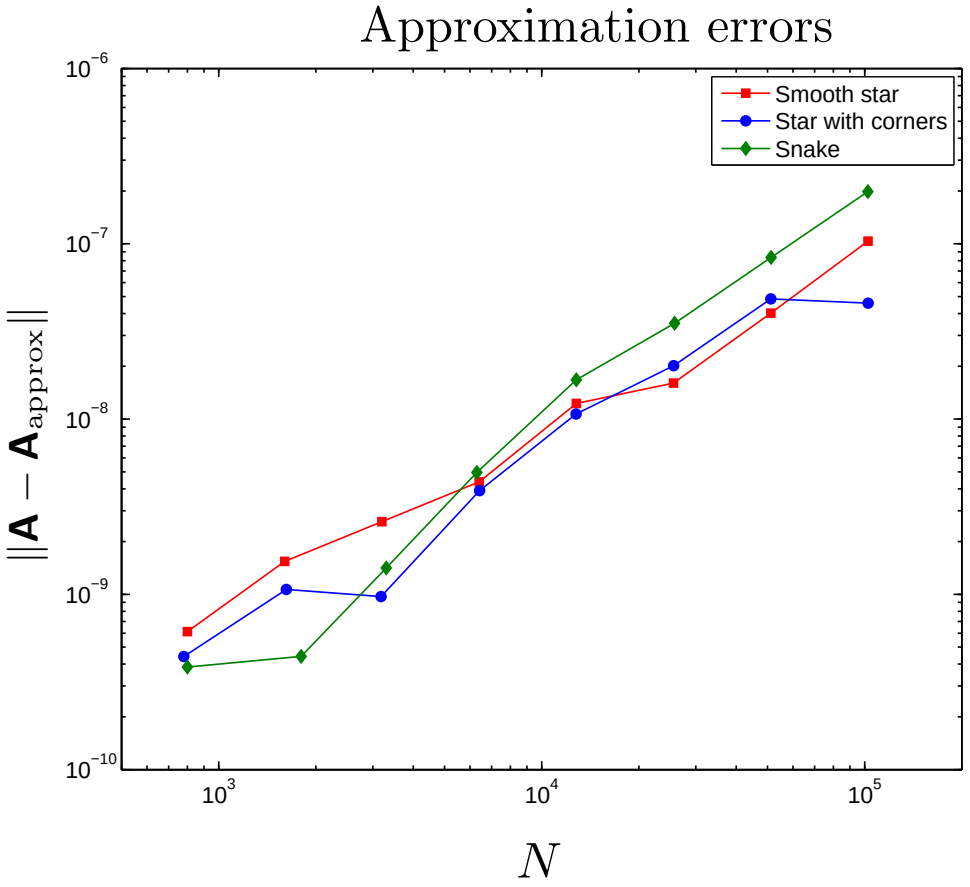


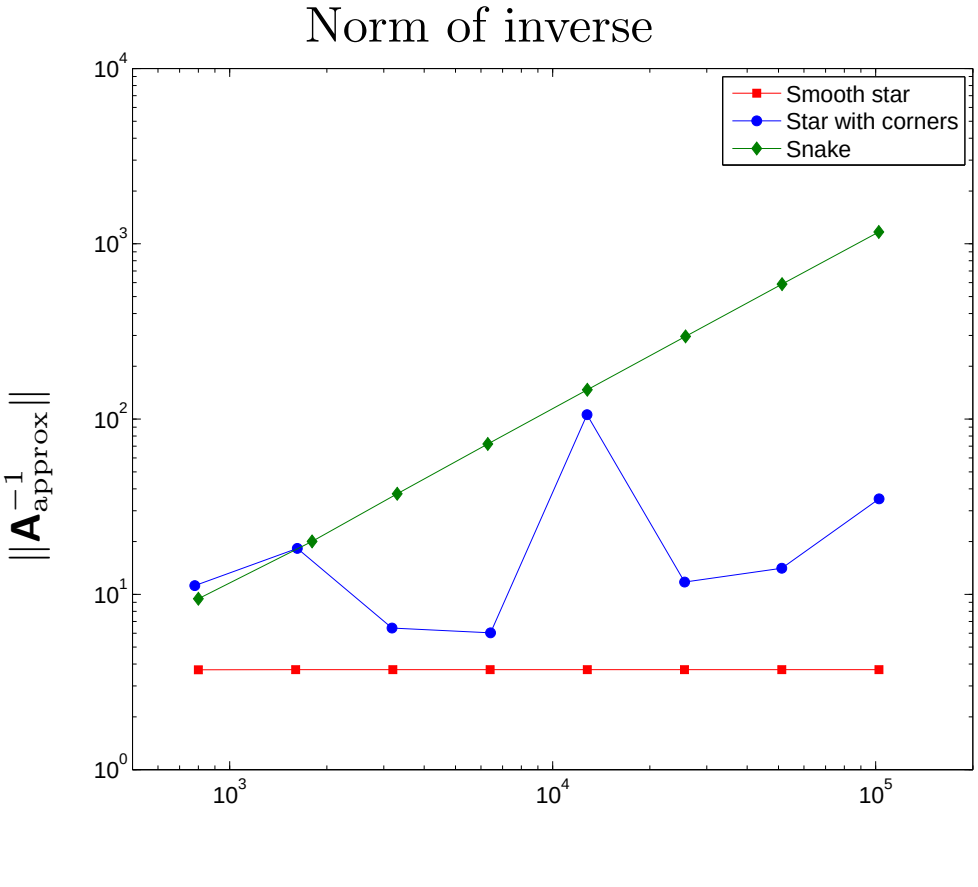
The graphs give the error in the approximation, and the forwards error in the inverse.

Within each graph, the four lines correspond to the four examples considered:

□ Smooth star ○ Star with corners ◇ Snake

1D numerical examples — BIEs in $\mathbb{R}^2$





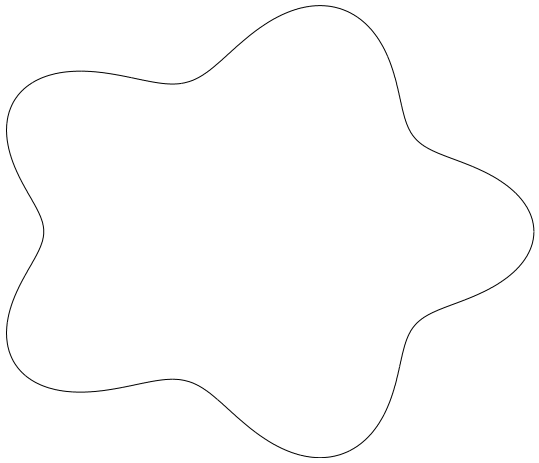
The graphs give the error in the approximation, and the norm of the inverse.

Within each graph, the four lines correspond to the four examples considered:

□ Smooth star ○ Star with corners ◇ Snake

1D numerical examples — BIEs in $\mathbb{R}^2$

Example: An interior Helmholtz Dirichlet problem



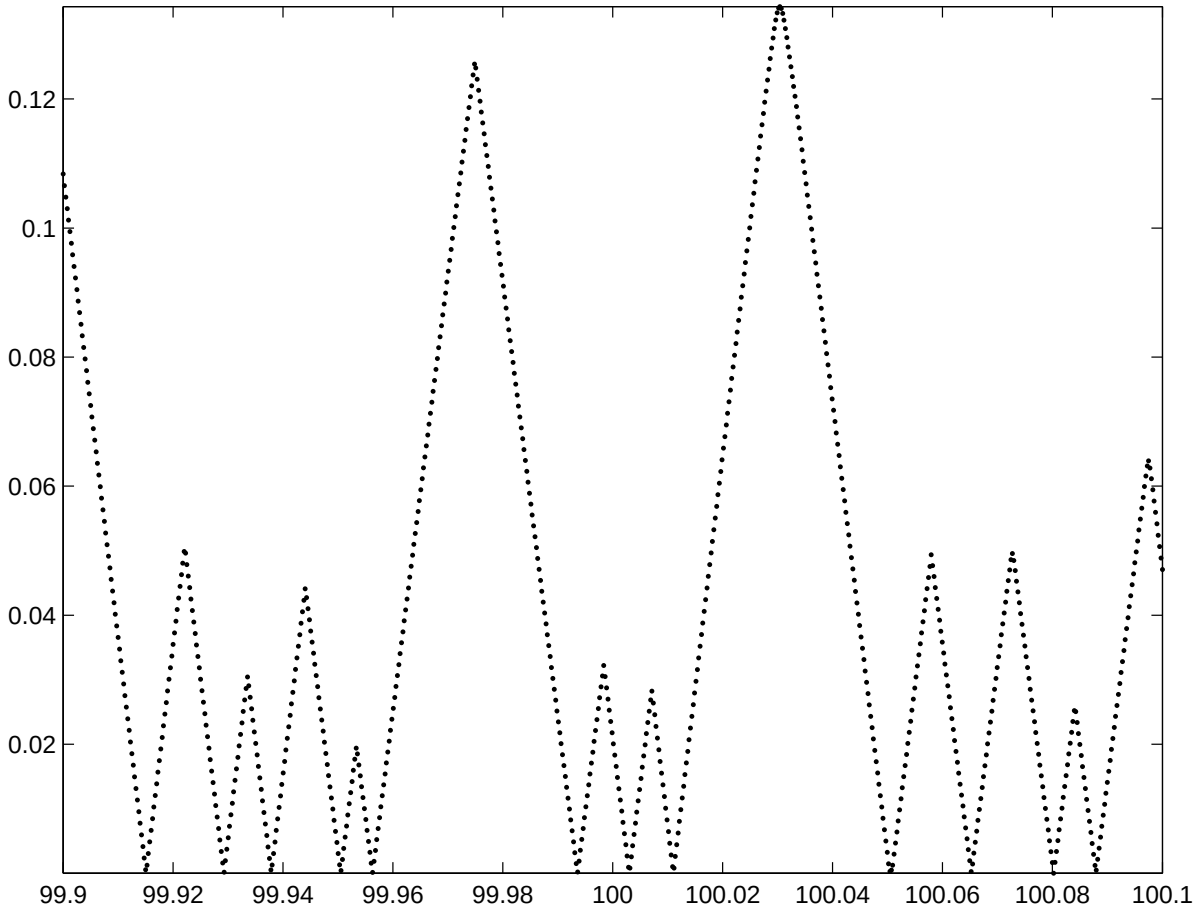
The diameter of the contour is about 2.5. An interior Helmholtz problem with Dirichlet boundary data was solved using $N = 6\,400$ discretization points, with a prescribed accuracy of 10^{-10}.

For $k = 100.011027569\cdots$, the smallest singular value of the boundary integral operator was $\sigma_{\min} = 0.00001366\cdots$.

Time for constructing the inverse: 0.7 seconds.

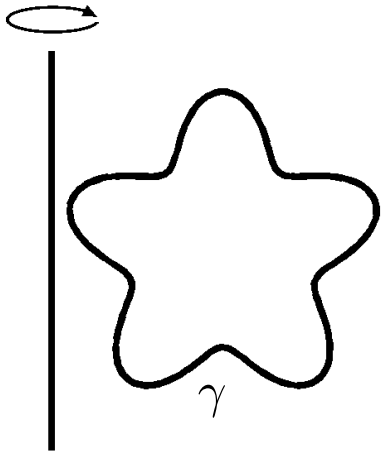
Error in the inverse: 10^{-5}.

1D numerical examples — BIEs in $\mathbb{R}^2$

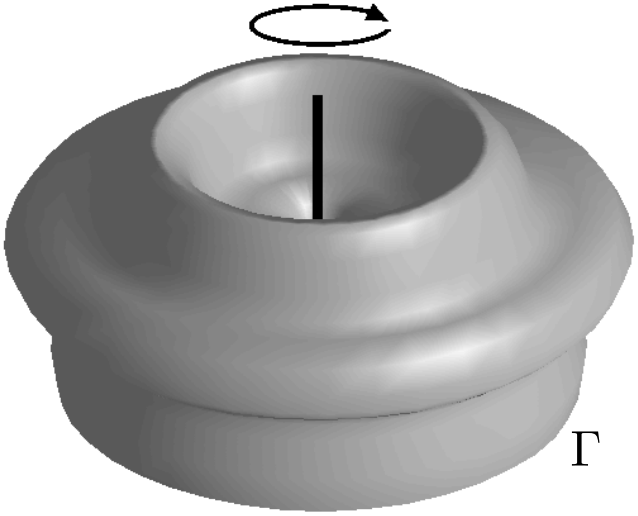


Plot of $\sigma_{\min}$ versus k for an interior Helmholtz problem on the smooth pentagram. The values shown were computed using a matrix of size $N = 6400$. Each point in the graph required about $60s$ of CPU time.

1D numerical examples — BIEs on rotationally symmetric surfaces



Generating curve



Surface

Let Γ be a surface of rotation generated by a curve γ, and consider a BIE associated with Laplace's equation:

$$(3) \qquad \frac{1}{2}\sigma(\mathbf{x}) + \int_\Gamma \frac{\mathbf{n}(\mathbf{y})\cdot(\mathbf{x}-\mathbf{y})}{4\pi|\mathbf{x}-\mathbf{y}|^3}\,\sigma(\mathbf{y})\,dA(\mathbf{y}) = f(\mathbf{x}). \quad \mathbf{x}\in\Gamma$$

To (3), we apply the Fourier transform in the azimuthal angle (executed computationally via the FFT) and get

$$\frac{1}{2}\sigma_n(\mathbf{x}) + \int_\gamma k_n(\mathbf{x},\mathbf{y})\,\sigma_n(\mathbf{y})\,dl(\mathbf{y}) = f_n(\mathbf{x}), \quad \mathbf{x}\in\gamma, \qquad n\in\mathbb{Z}.$$

Then discretize the sequence of equations on γ using the direct solvers described (with special quadratures, *etc*).

We discretized the surface using 400 Fourier modes, and 800 points on γ for a total problem size of

$$N = 320\,000.$$

For typical loads, the relative error was less than 10^{-10} and the CPU times were

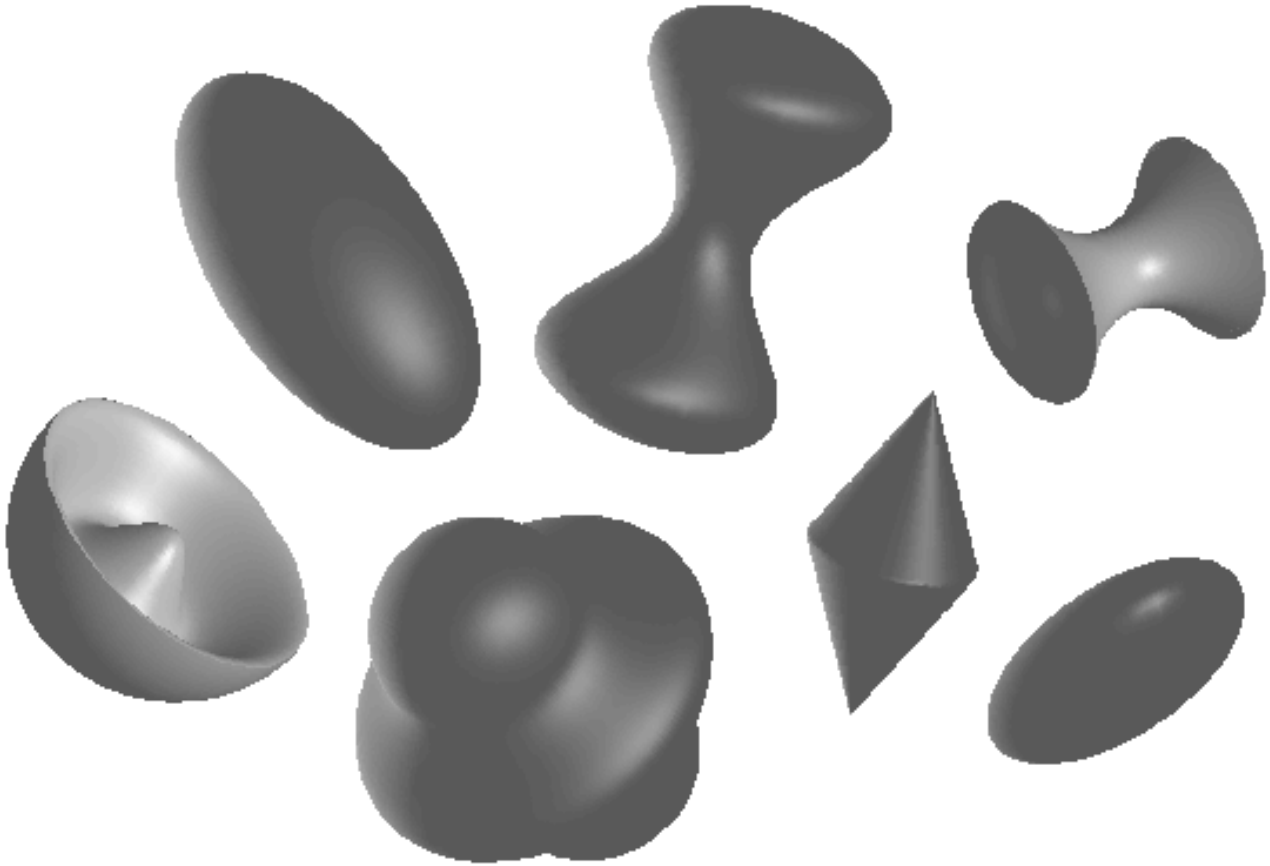
$$T_{\text{invert}} = 2\text{min} \qquad T_{\text{solve}} = 0.3\text{sec}.$$

1D numerical examples — BIEs on rotationally symmetric surfaces

Work in progress (with Sijia Hao): Extension to multibody acoustic scattering:

Individual scattering matrices are constructed via a relatively expensive pre-computation.

Inter-body interactions are handled via the wideband FMM and an iterative solver.

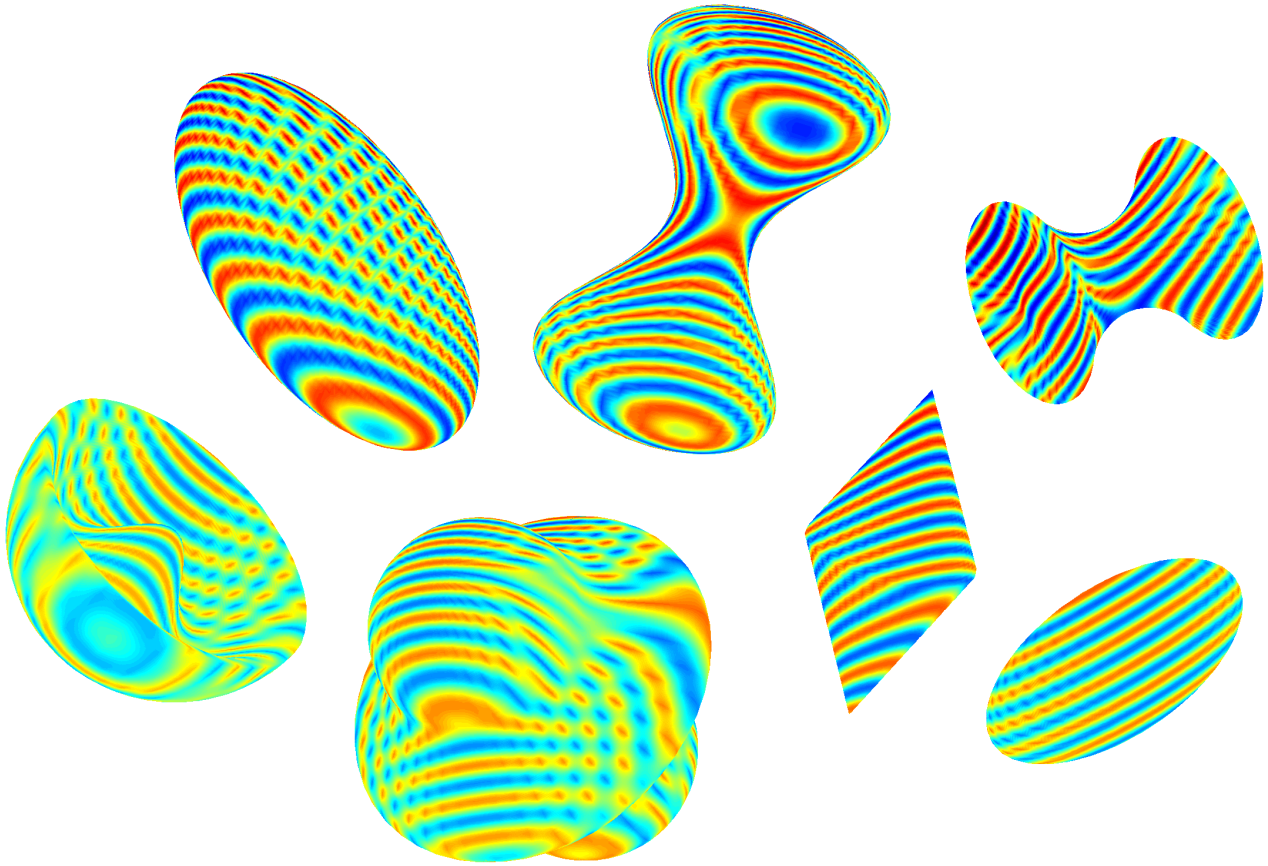


1D numerical examples — BIEs on rotationally symmetric surfaces

Work in progress (with Sijia Hao): Extension to multibody acoustic scattering:

Individual scattering matrices are constructed via a relatively expensive pre-computation.

Inter-body interactions are handled via the wideband FMM and an iterative solver.

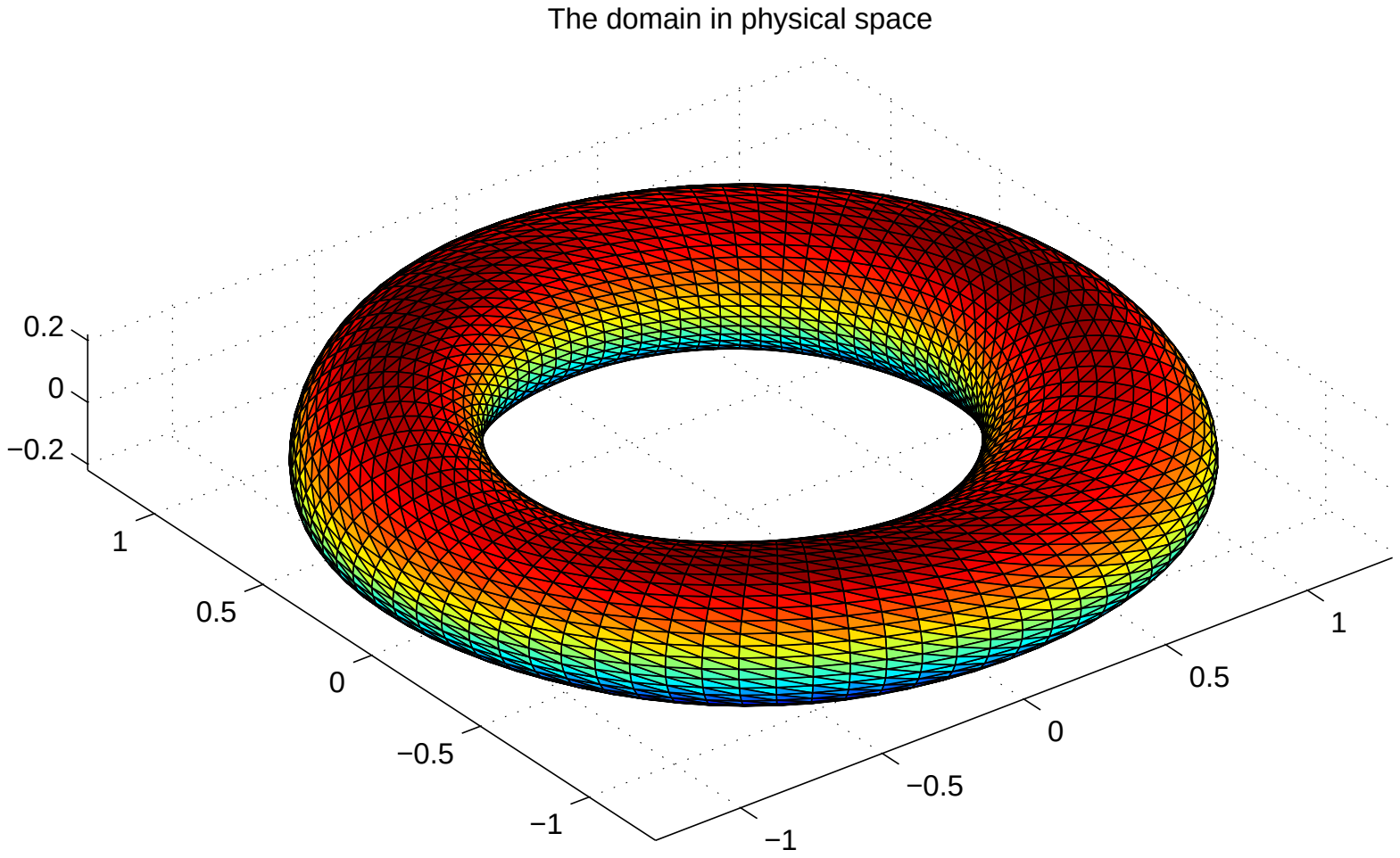


Extension to problems on 2D domain

As a model problem we consider a single layer potential on a **deformed** torus:

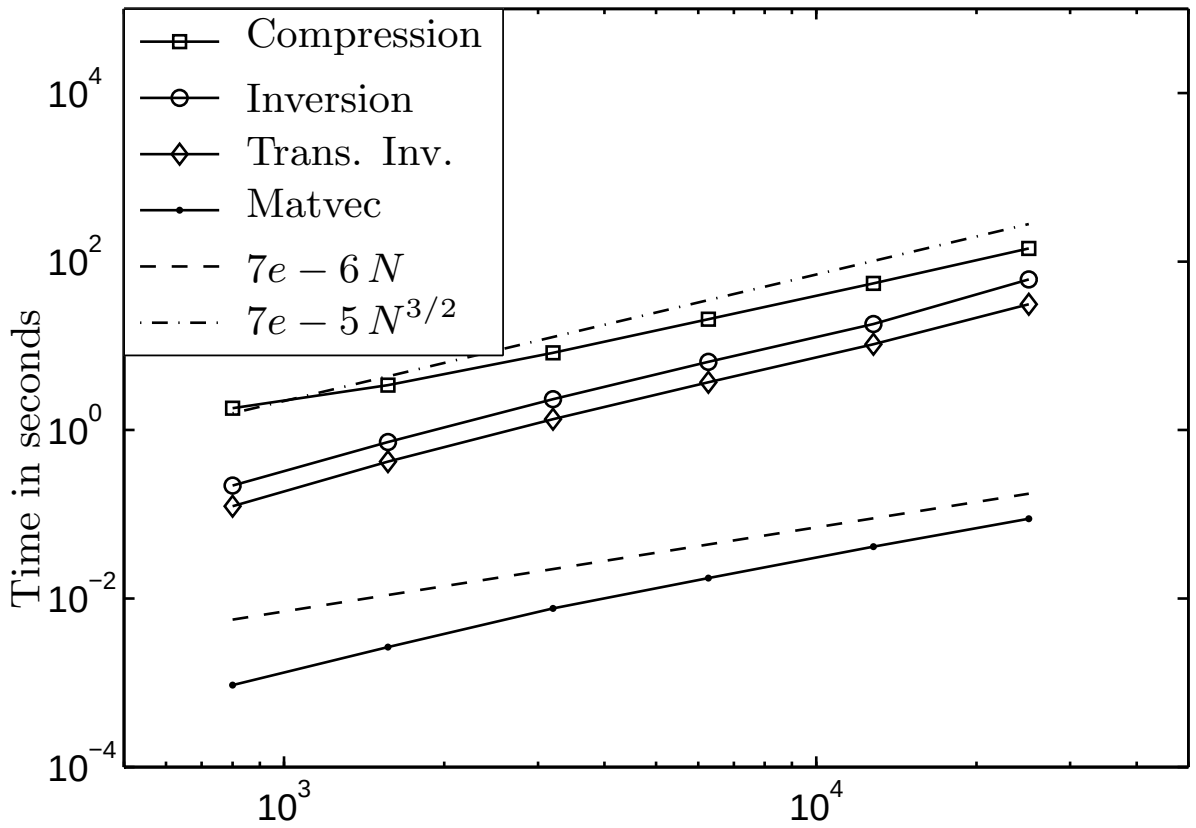
$$[A\sigma](x) = \sigma(x) + \int_\Gamma \log|x-y|\,\sigma(y)\,dA(y), \qquad y \in \Gamma,$$

where Γ is the domain



*This is **not** rotationally symmetric.*

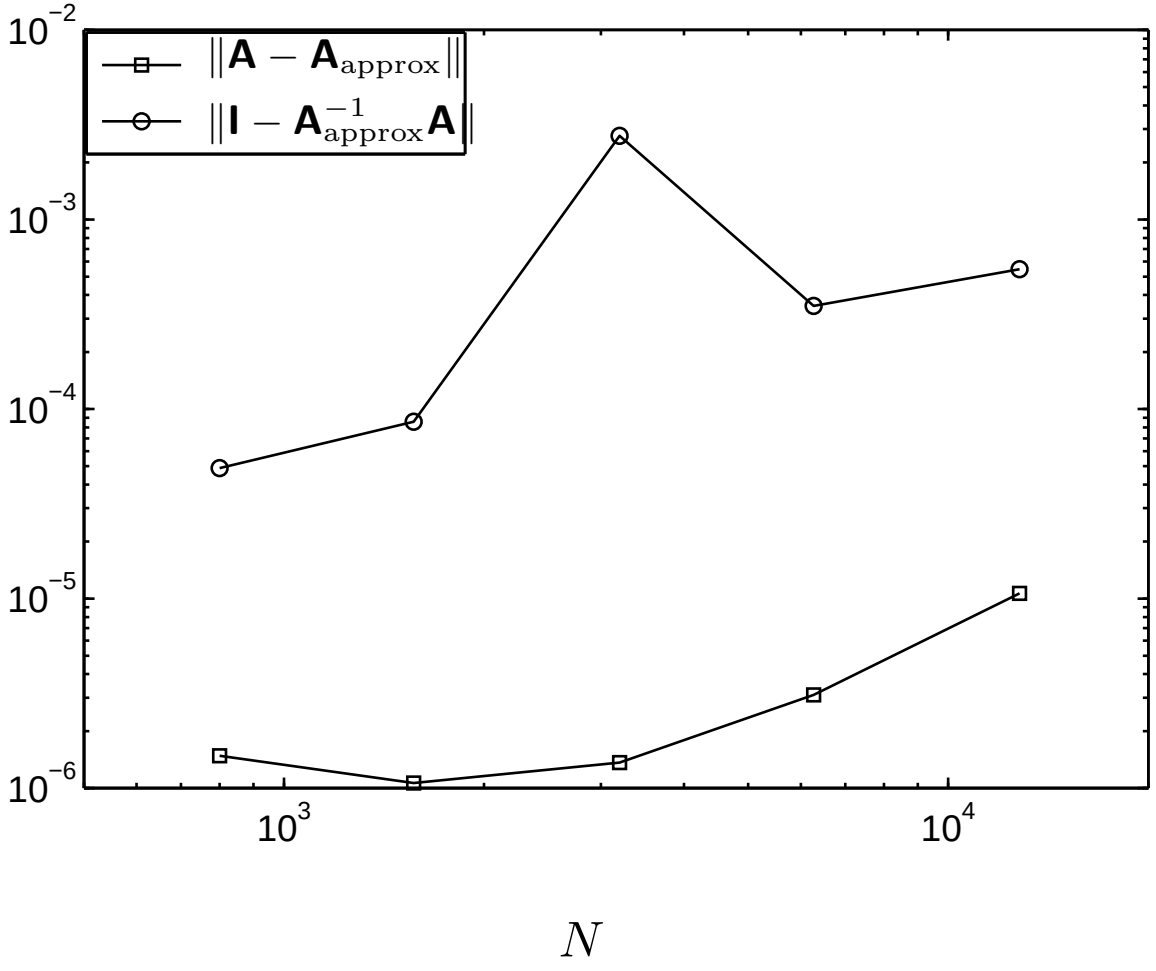
Performance of direct solver for the torus domain



Observe that for a BIE with $N = 25\,600$, the inverse can be applied in 0.09 seconds.

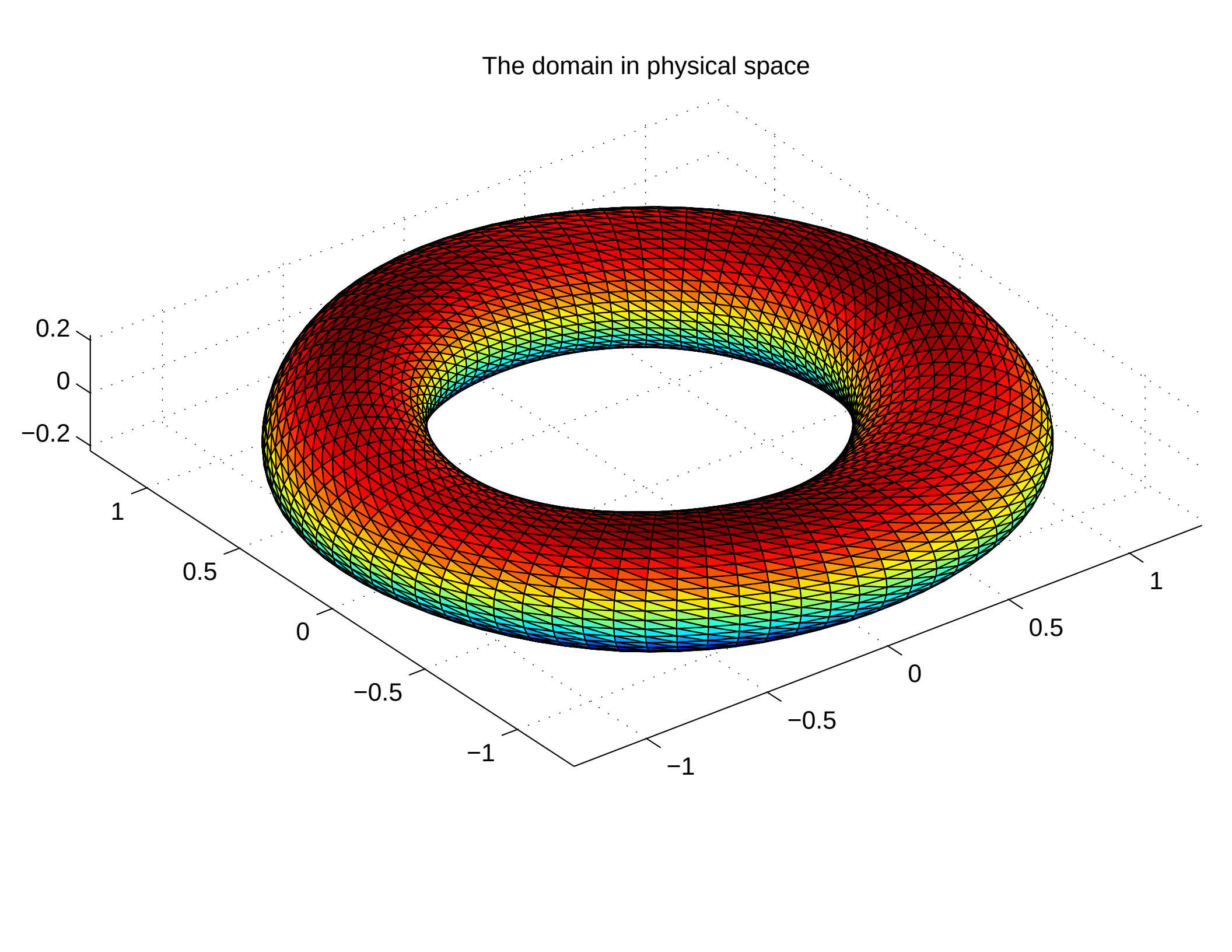
The asymptotic complexity is:

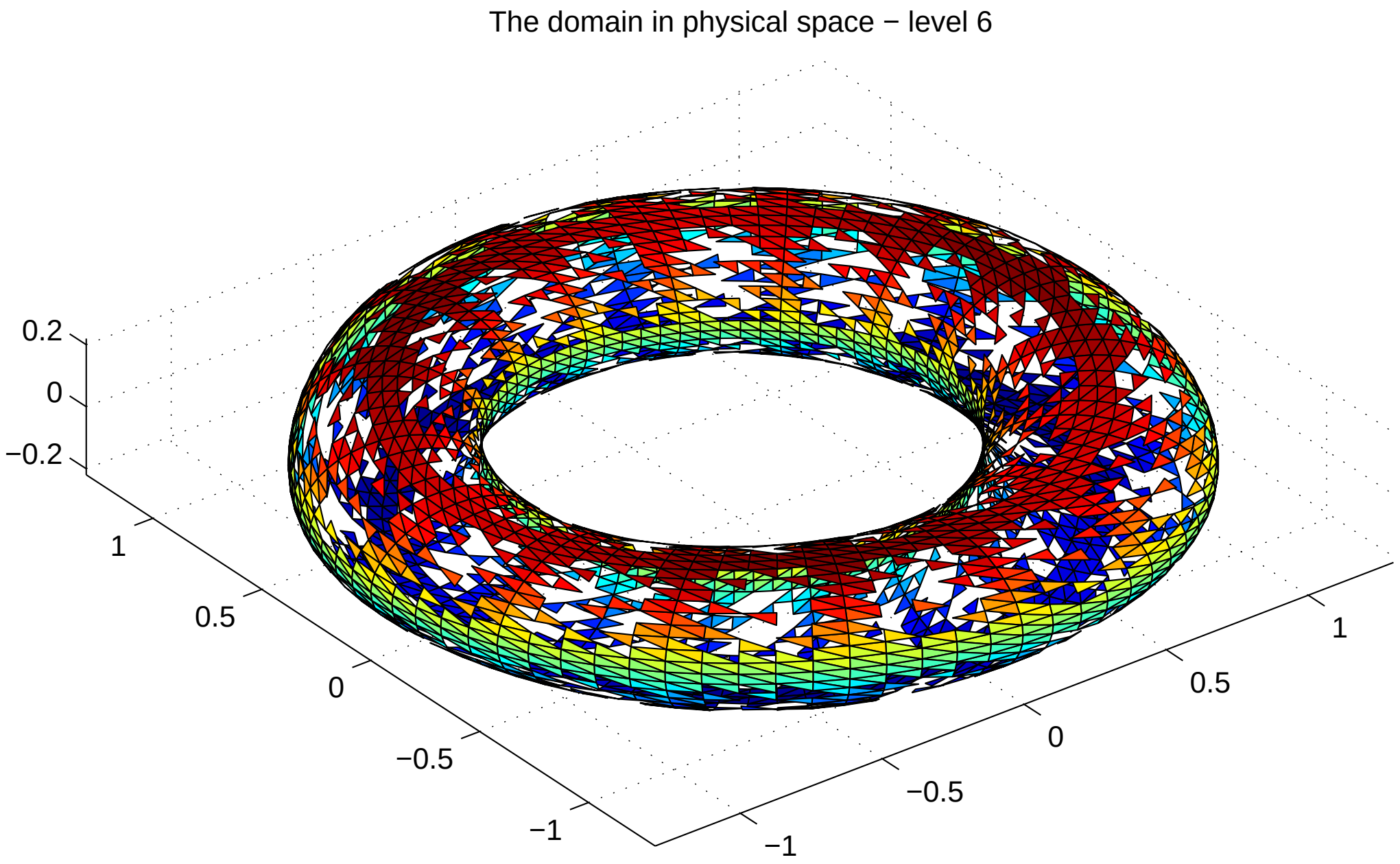
Inversion step:	$O(N^{1.5})$ (with small scaling constant)
Application of the inverse:	$O(N)$



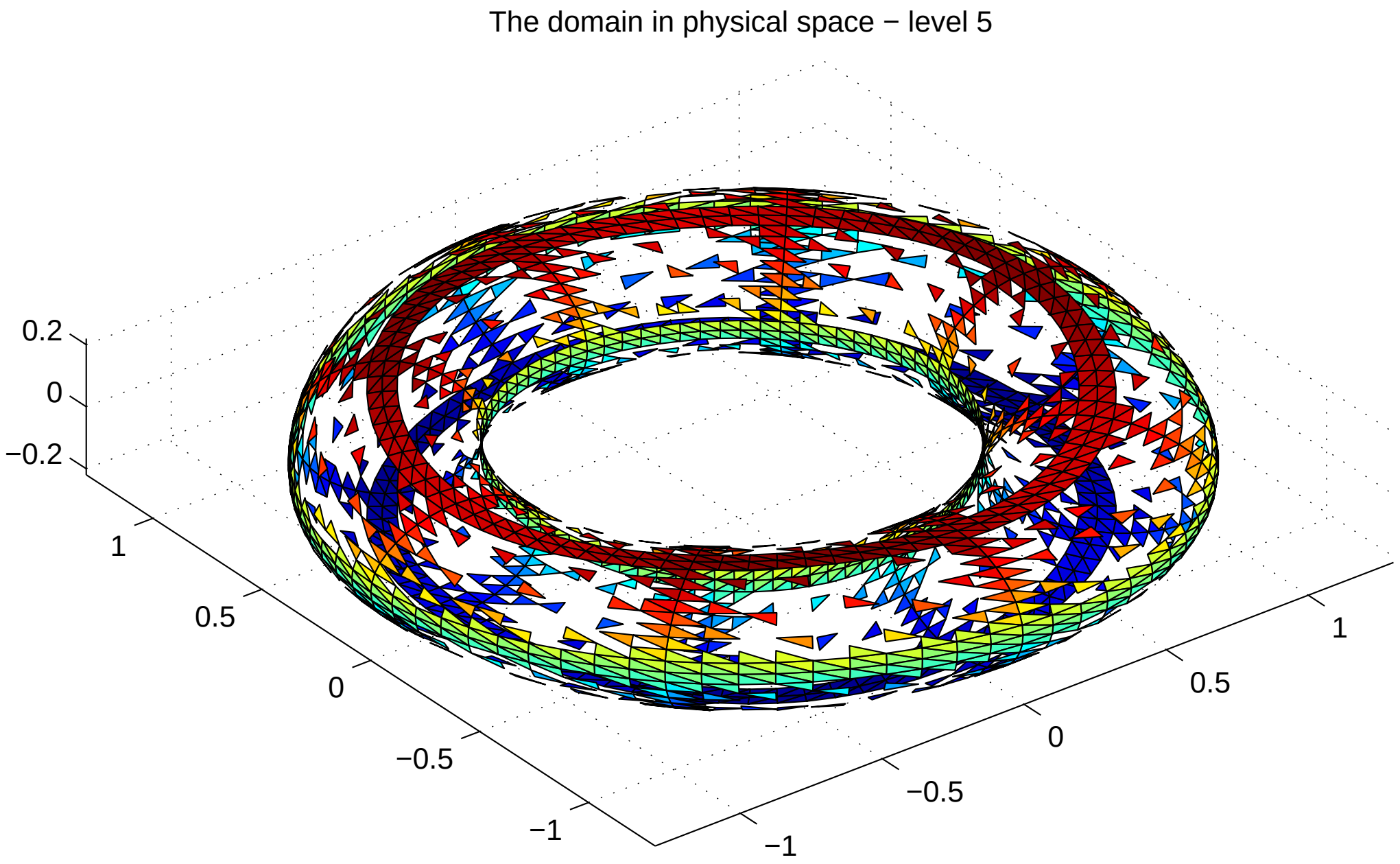
Errors for the same problem as the previous slide.

A (useful) curious fact: The inversion procedure computationally constructs a Nyström discretization of the domain at each of the levels.

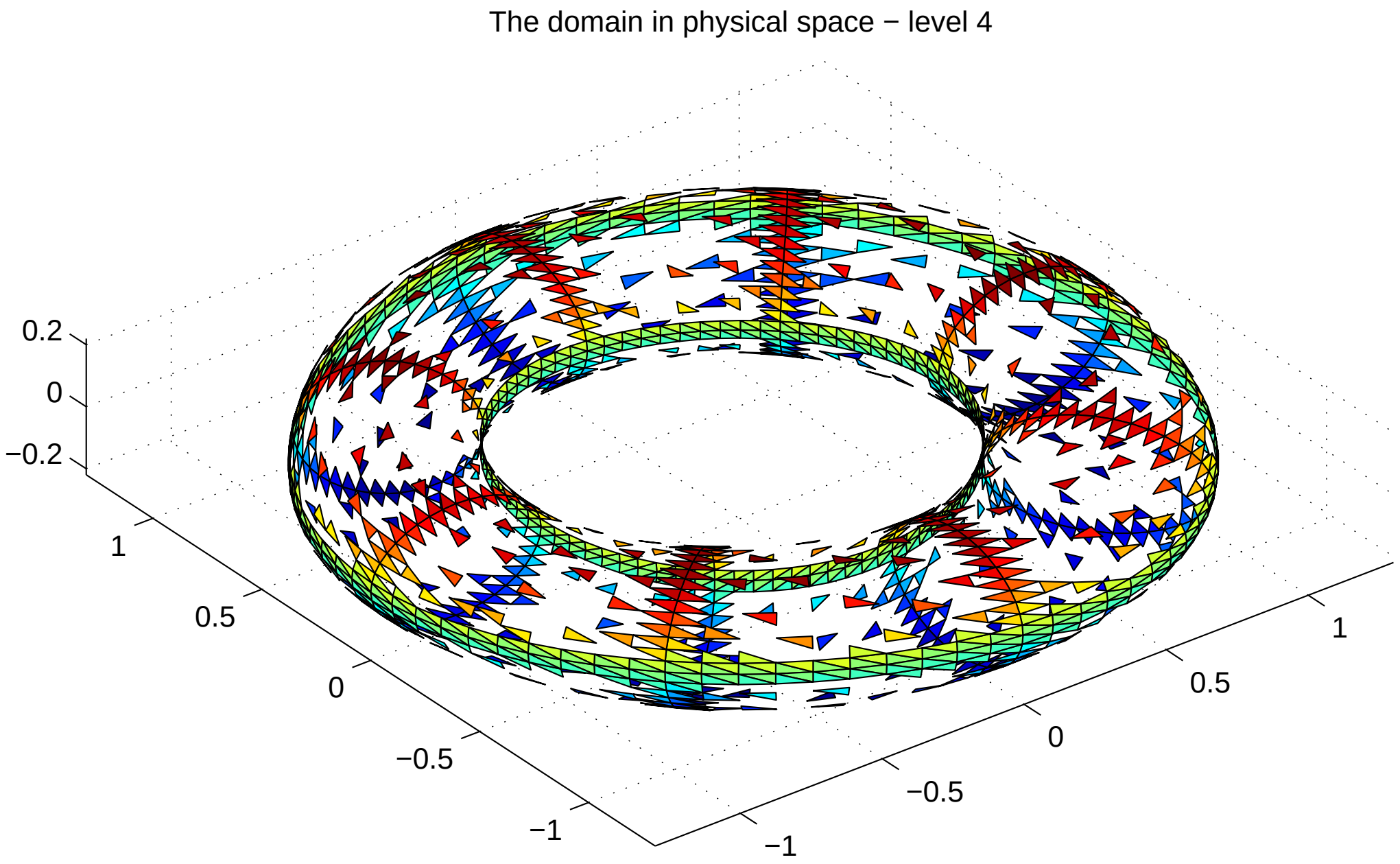

The domain in physical space
0.2
0
−0.2
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1



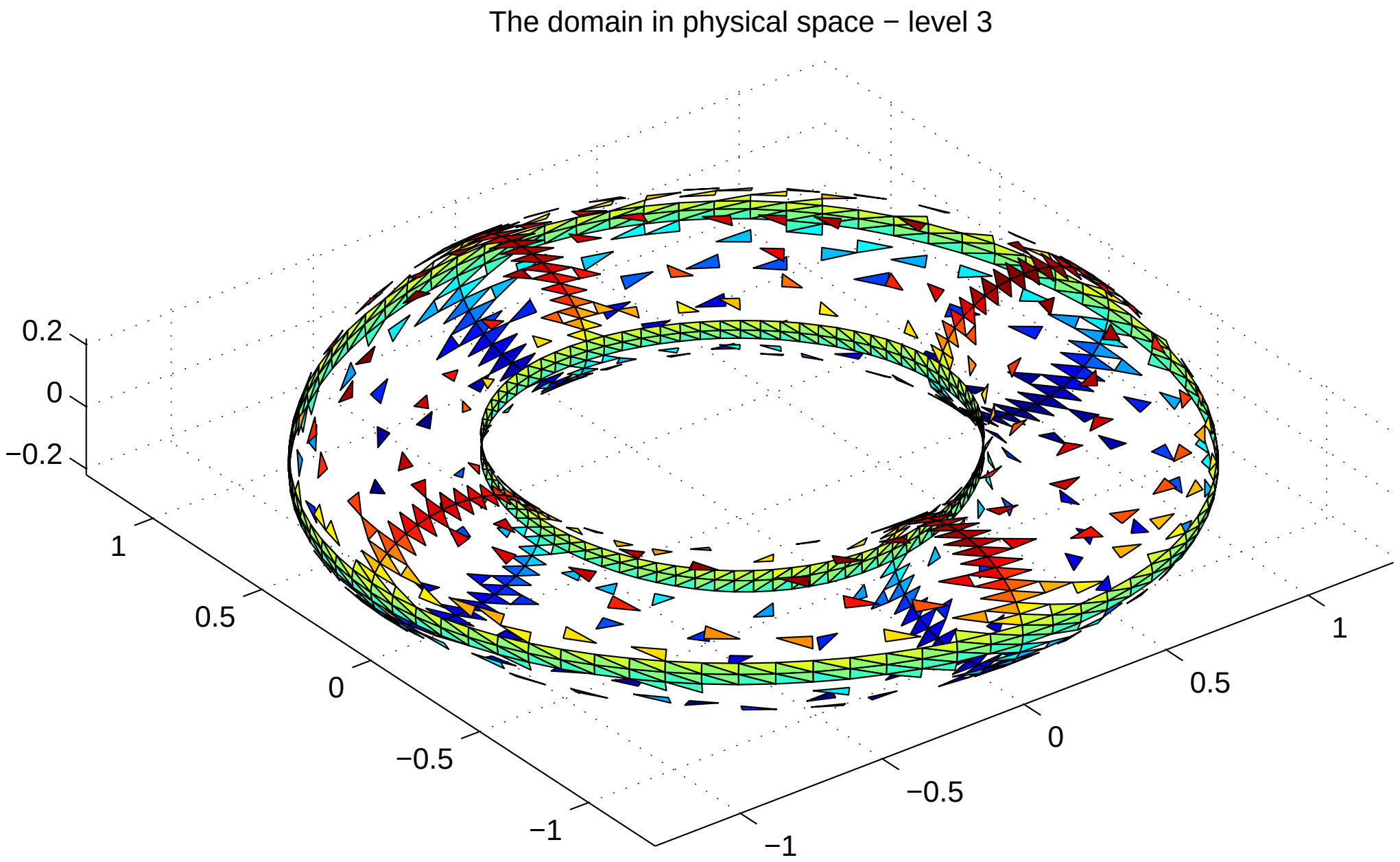
The reduced matrix represents a Nyström discretization supported on the panels shown.



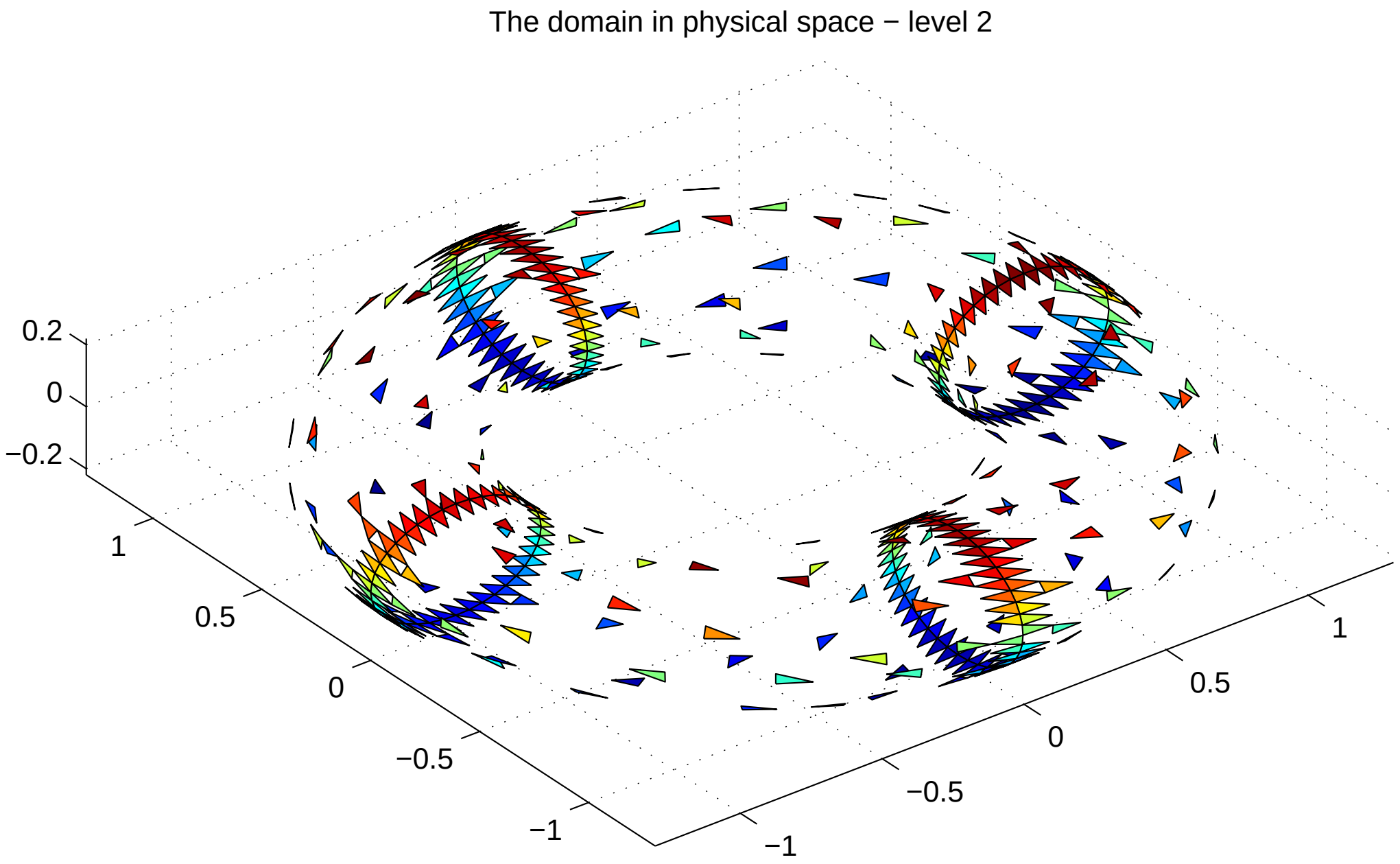
The reduced matrix represents a Nyström discretization supported on the panels shown.



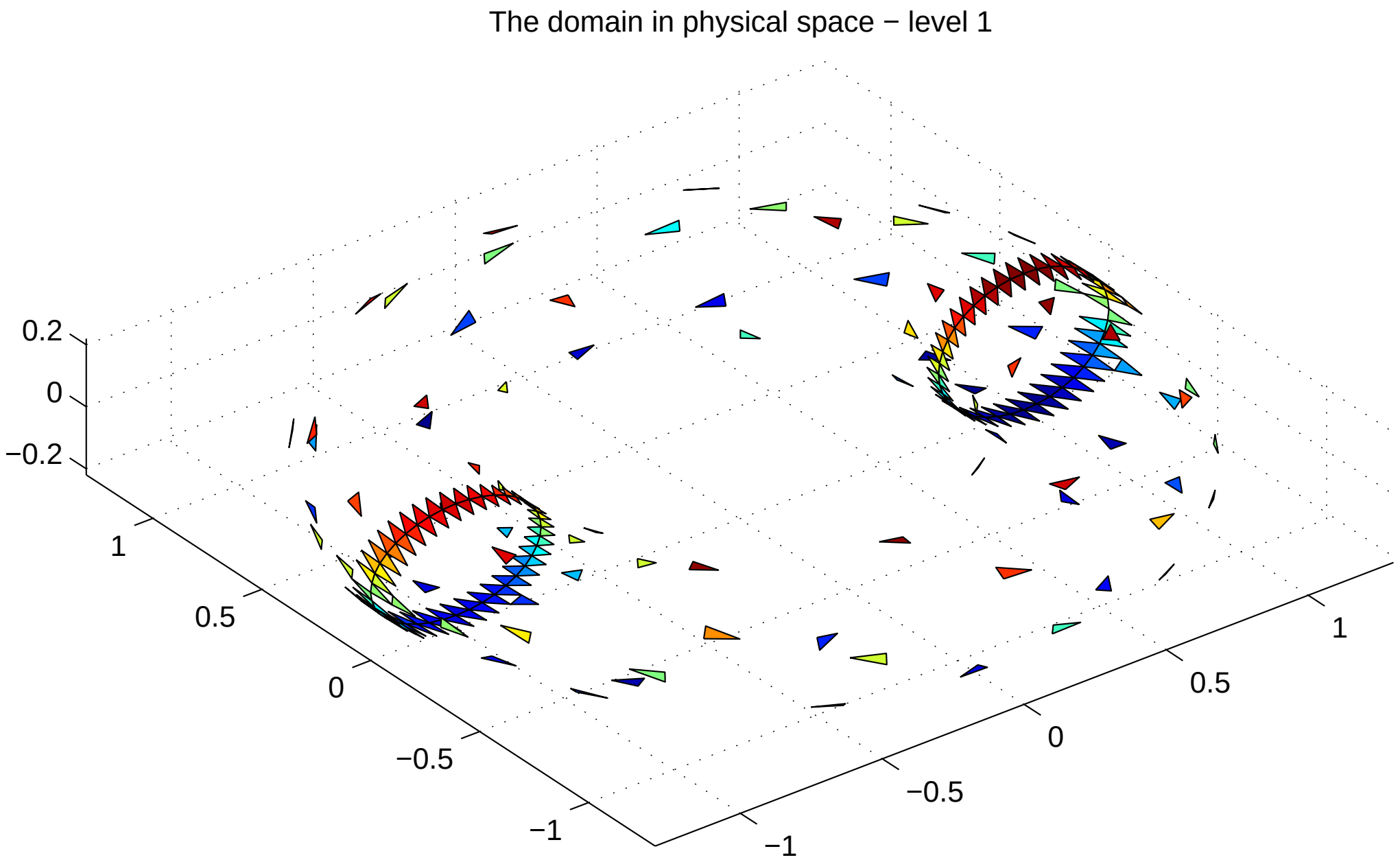
The reduced matrix represents a Nyström discretization supported on the panels shown.



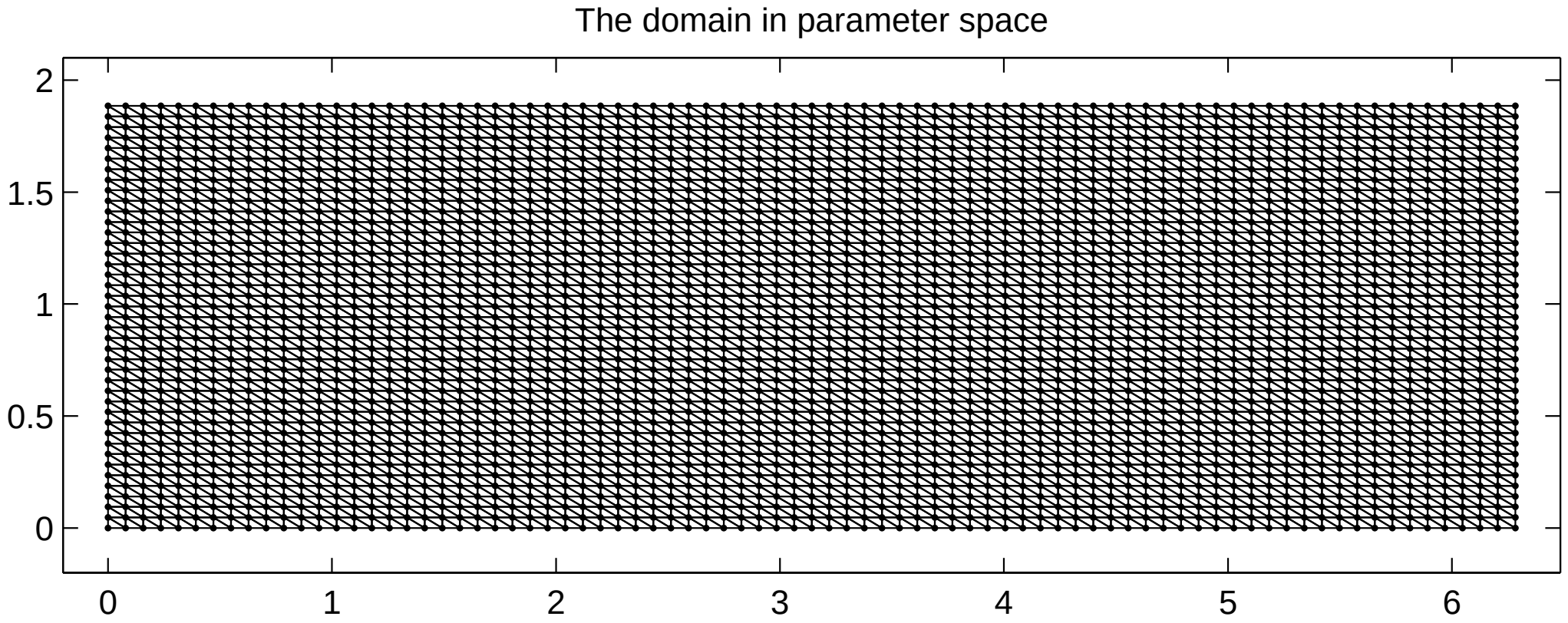
The reduced matrix represents a Nyström discretization supported on the panels shown.



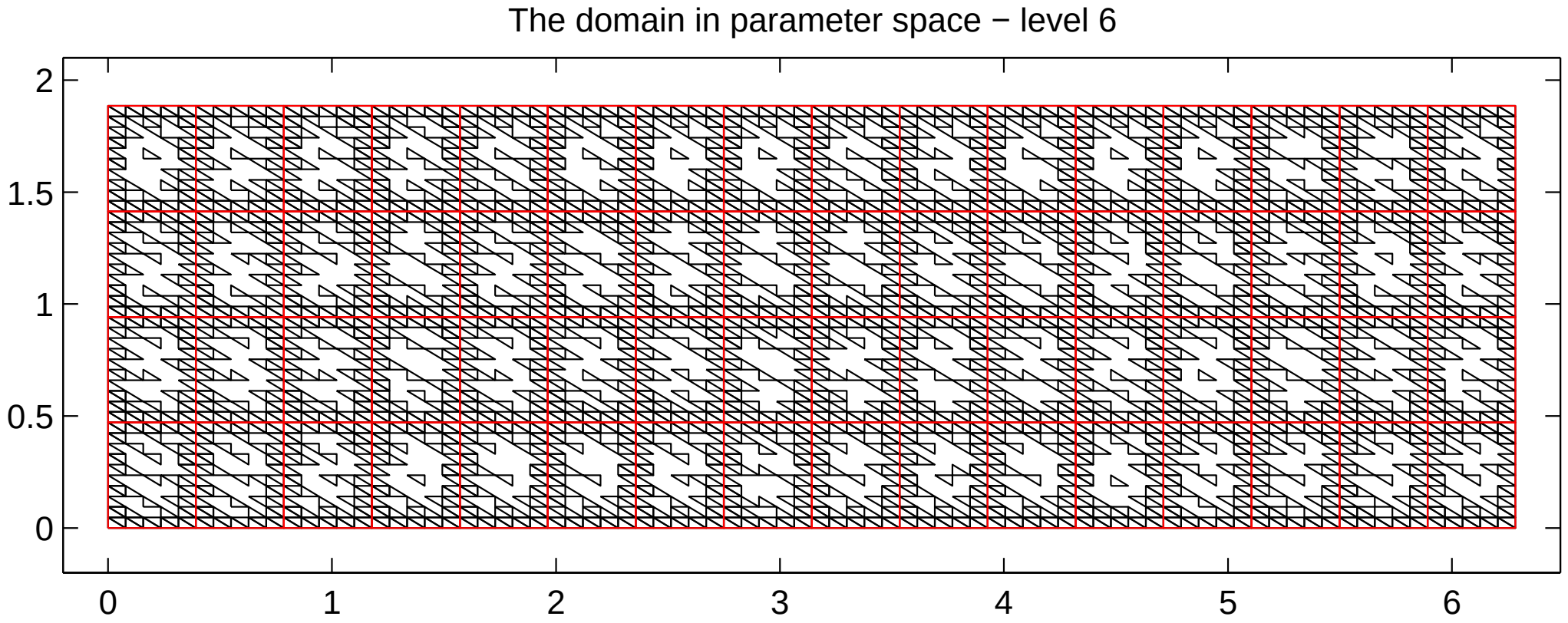
The reduced matrix represents a Nyström discretization supported on the panels shown.



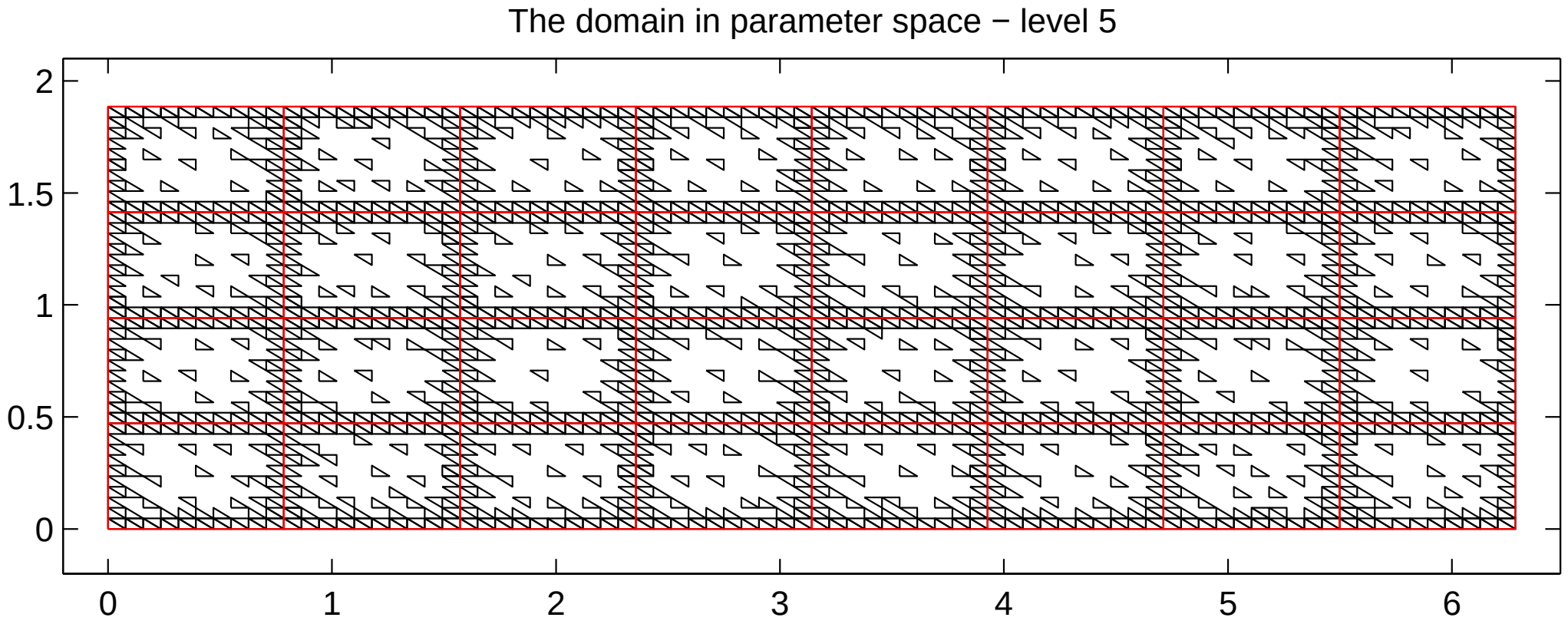
The reduced matrix represents a Nyström discretization supported on the panels shown.



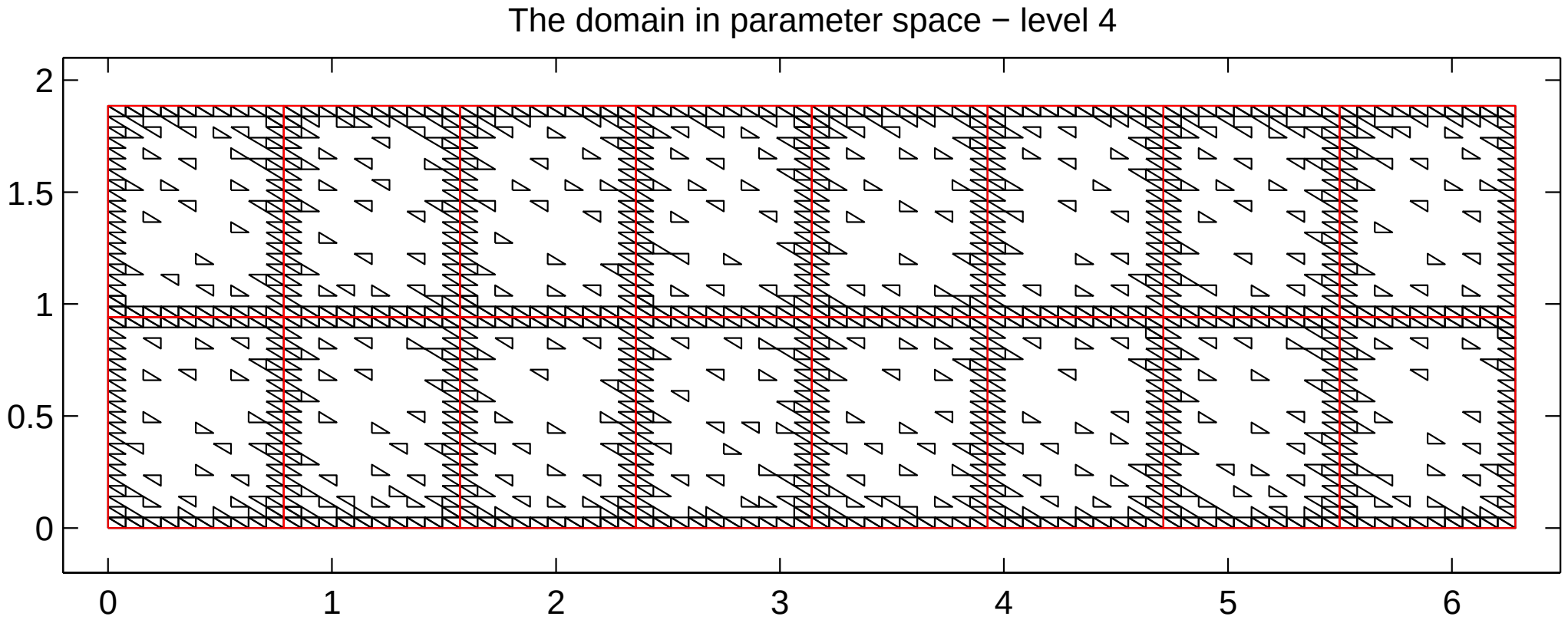
The reduced matrix represents a Nyström discretization supported on the panels shown.



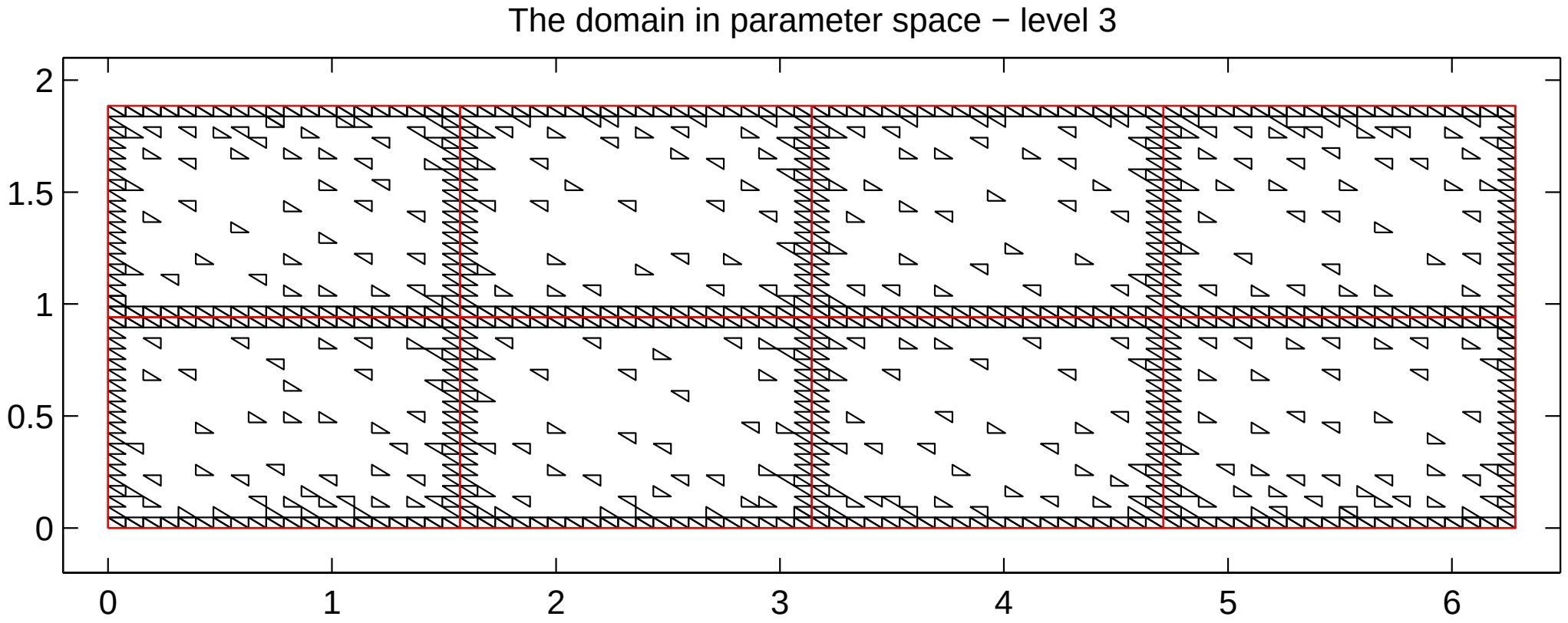
The reduced matrix represents a Nyström discretization supported on the panels shown.



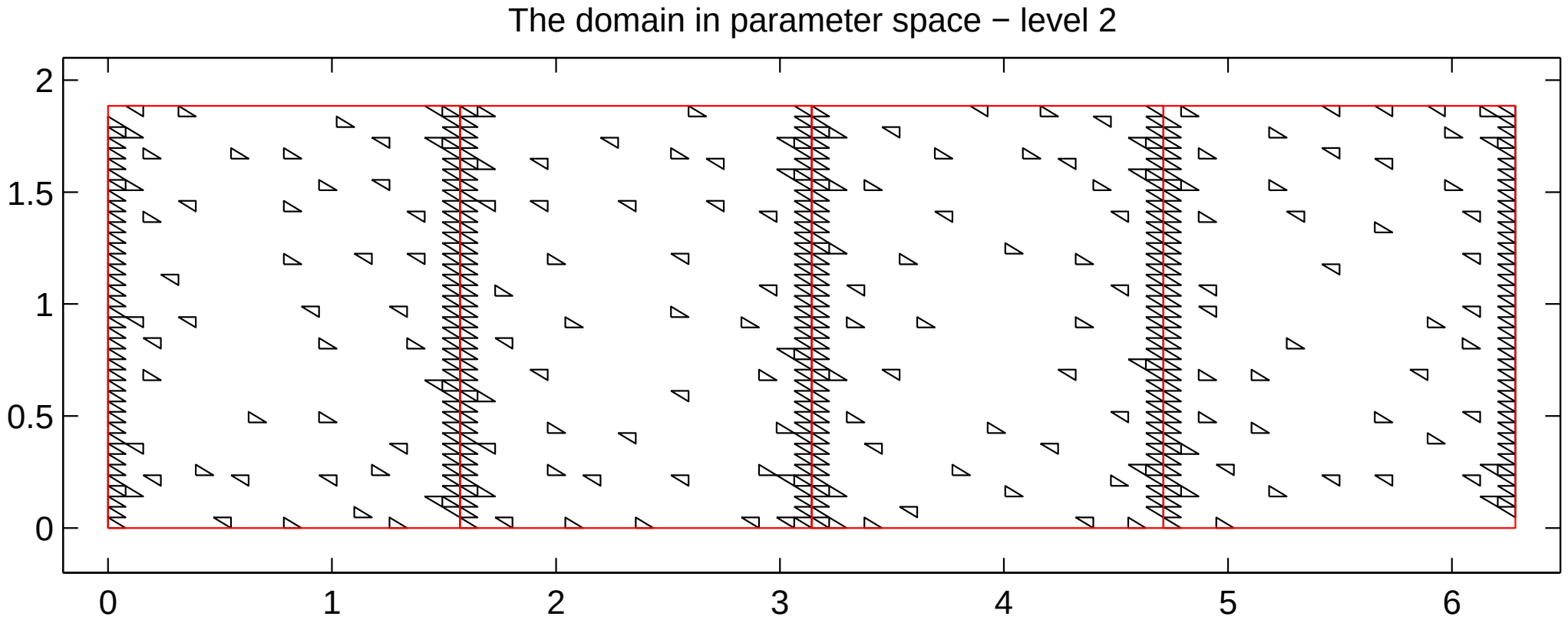
The reduced matrix represents a Nyström discretization supported on the panels shown.



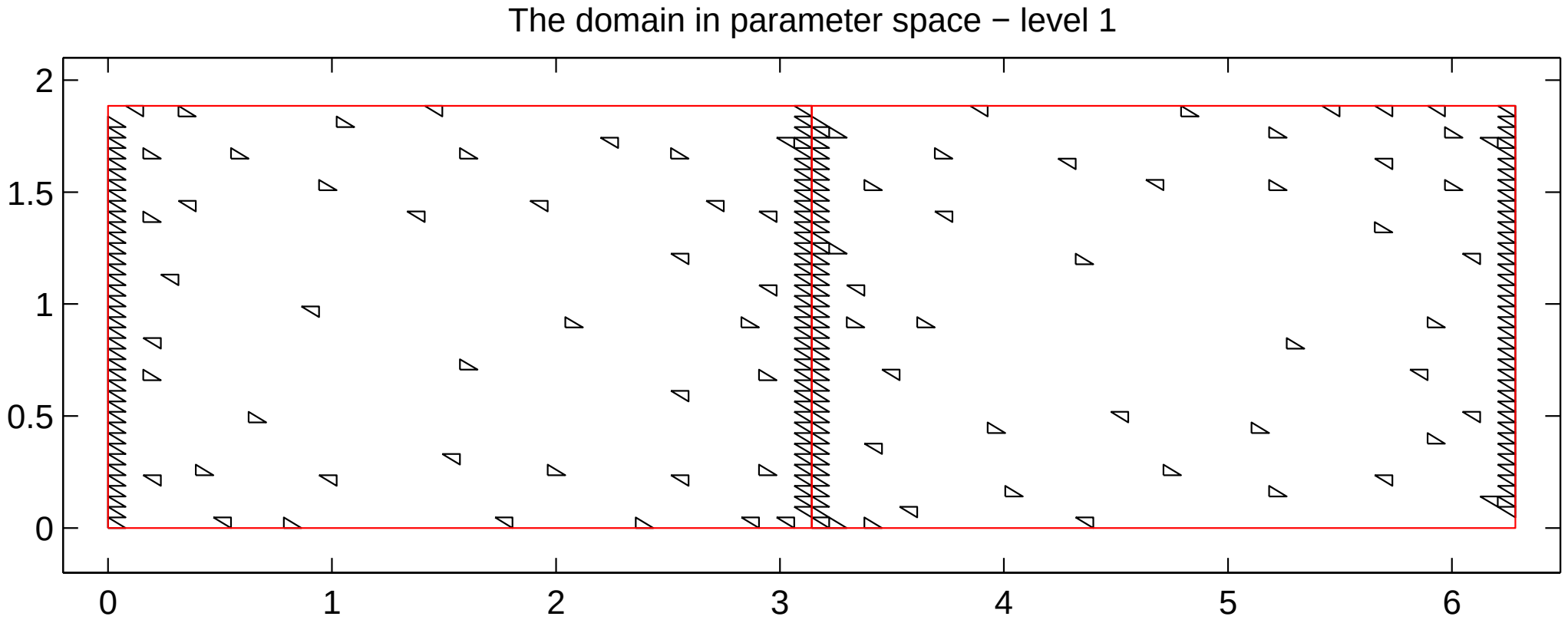
The reduced matrix represents a Nyström discretization supported on the panels shown.



The reduced matrix represents a Nyström discretization supported on the panels shown.



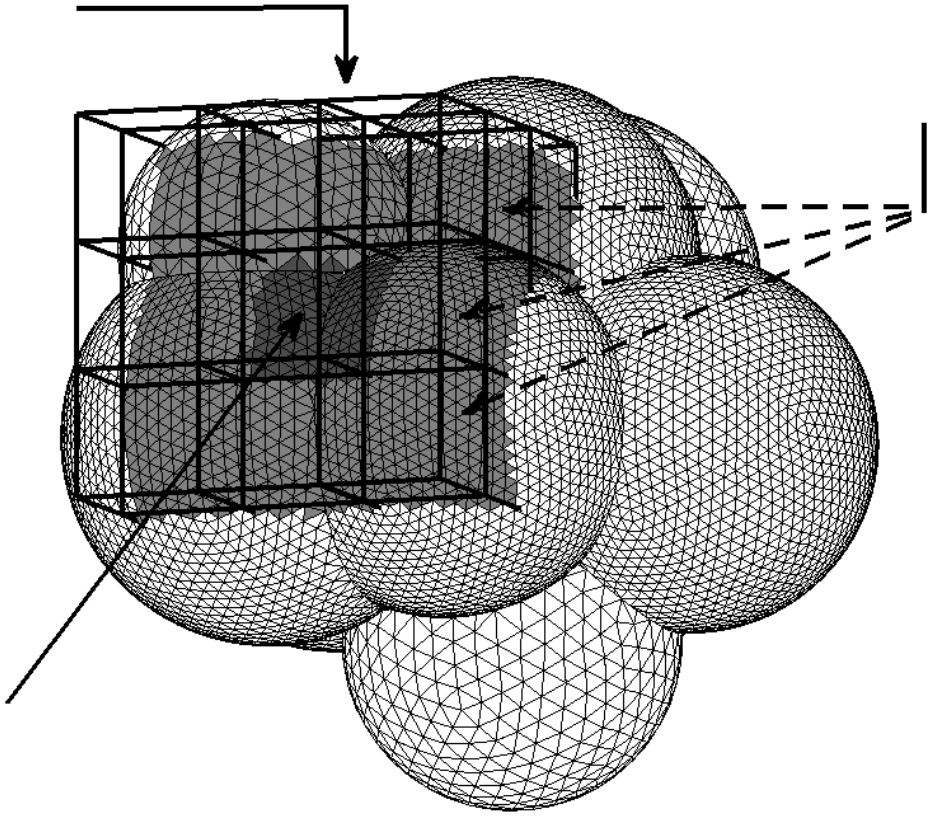
The reduced matrix represents a Nyström discretization supported on the panels shown.



The reduced matrix represents a Nyström discretization supported on the panels shown.

The code for the torus domain was based on a binary tree that partitioned the domain in *parameter space.* This leads to high efficiency when it works, but is not very generic.

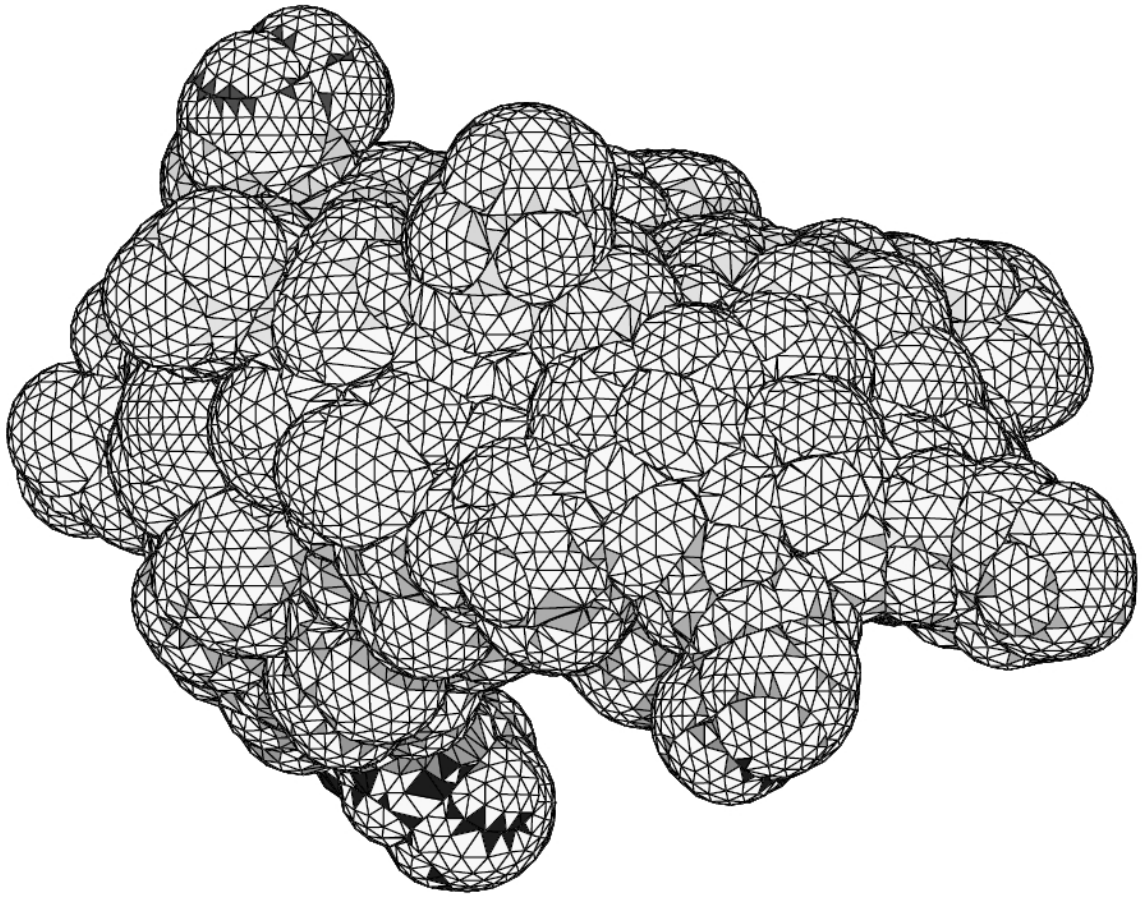
For a more robust code, we instead construct a binary tree in *physical space:*



For "regular" surfaces, the resulting oct-tree is sparsely populated, and interactions rank max out at $O(N^{1/2})$. The complexity therefore remains:

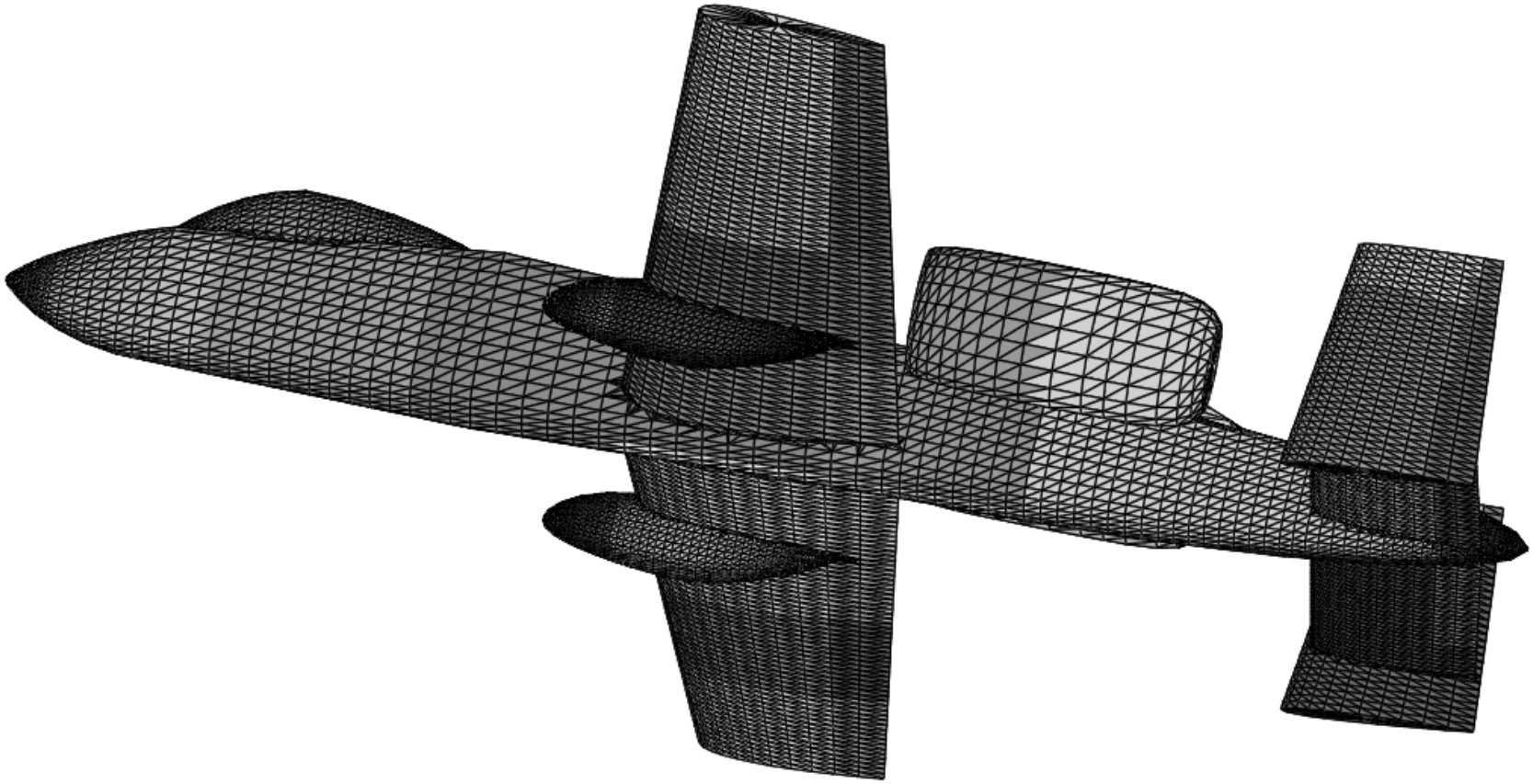
Inversion step:	$O(N^{1.5})$ (with small scaling constant)
Application of the inverse:	$O(N)$

Such a code has been implemented and tested on model problems such as molecular surfaces and aircraft fuselages.



Example: Triangulated aircraft

Computation carried out by Denis Gueyffier at Courant.

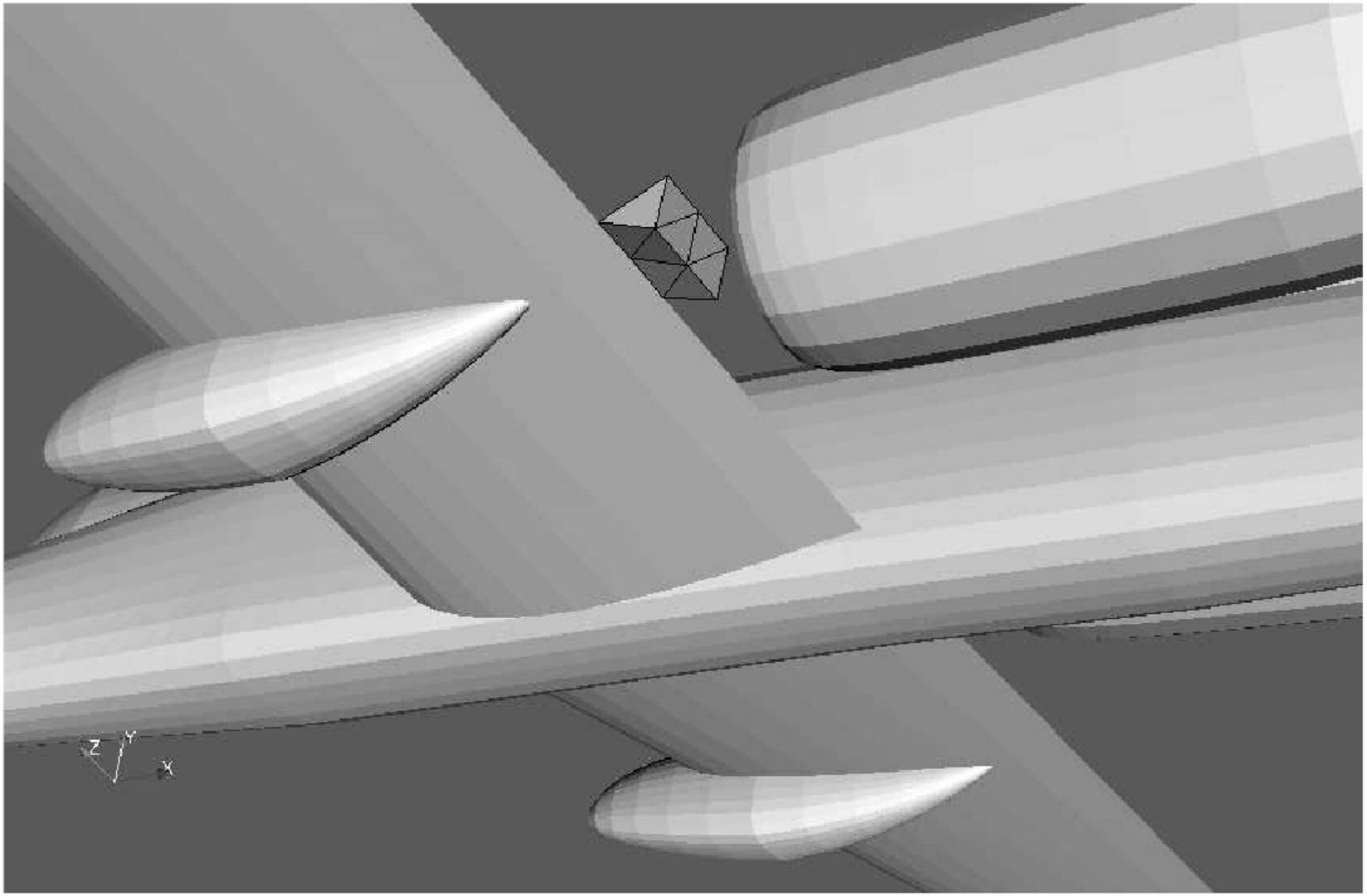


Laplace's equation. 28 000 triangles. Standard office desktop.

Cost of very primitive inversion scheme (low accuracy, etc.): 15 min

Cost of applying the inverse: 0.2 sec

From *Fast direct solvers for integral equations in complex three-dimensional domains*, by Greengard, Gueyffier, Martinsson, Rokhlin, Acta Numerica 2009.



Observation: Local updates to the geometry are very cheap. Adding a (not so very aerodynamic) flap corresponds to a rank-15 update and can be done in a fraction of a second.

Note: While our codes are very primitive at this point, there exist extensive $\mathcal{H}/\mathcal{H}^2$-matrix based libraries with better asymptotic estimates for inversion. `www.hlib.org`

Comments on "fast" direct solvers for BIEs in $\mathbb{R}^3$

The cost of applying a computed inverse is excellent — a fraction of a second even for problems with 10^5 or so degrees of freedom.

Storage requirements are acceptable — $O(N)$ with a modest constant of scaling.

The cost of the inversion/factorization is $O(N^{1.5})$ and is not entirely satisfactory.
Can it be reduced to $O(N)$ or $O(N \log N)$?

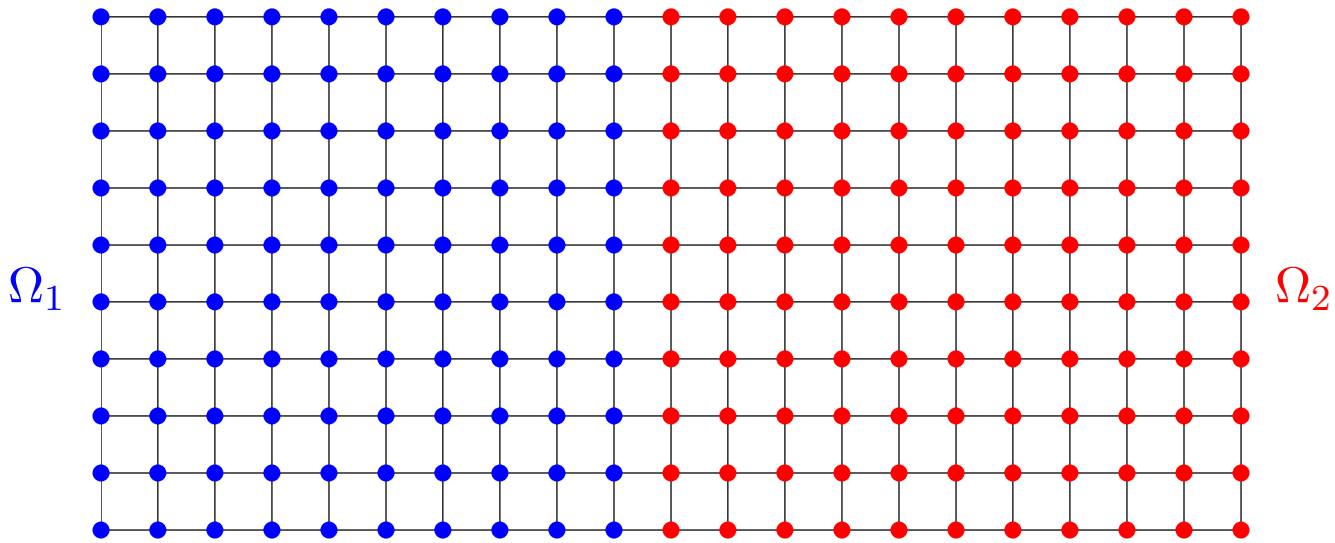
Recall that the problem is that the interaction of a block with m discretization points scales as $O(m^{0.5})$ as m grows. It is inversion/factorization/matrix-matrix multiplications of dense matrices of size $O(m^{0.5}) \times O(m^{0.5})$ that bring us down. (In the "1D" case, these matrices were of size $O(\log m) \times O(\log m)$.)

Cure: It turns out these dense matrices are themselves HSS matrices!

Codes exploiting this fact to construct linear complexity direct solvers for surface BIEs are currently being developed.

The "recursion on dimension" method described is very similar to *nested dissection* methods for inverting/factoring sparse matrices arising from the finite element or finite difference discretization of an elliptic PDE.

To illustrate, consider a square regular grid for the five-point stencil on a rectangular domain with $N = 2n \times n$ grid points.



The coefficient matrix can be split as before

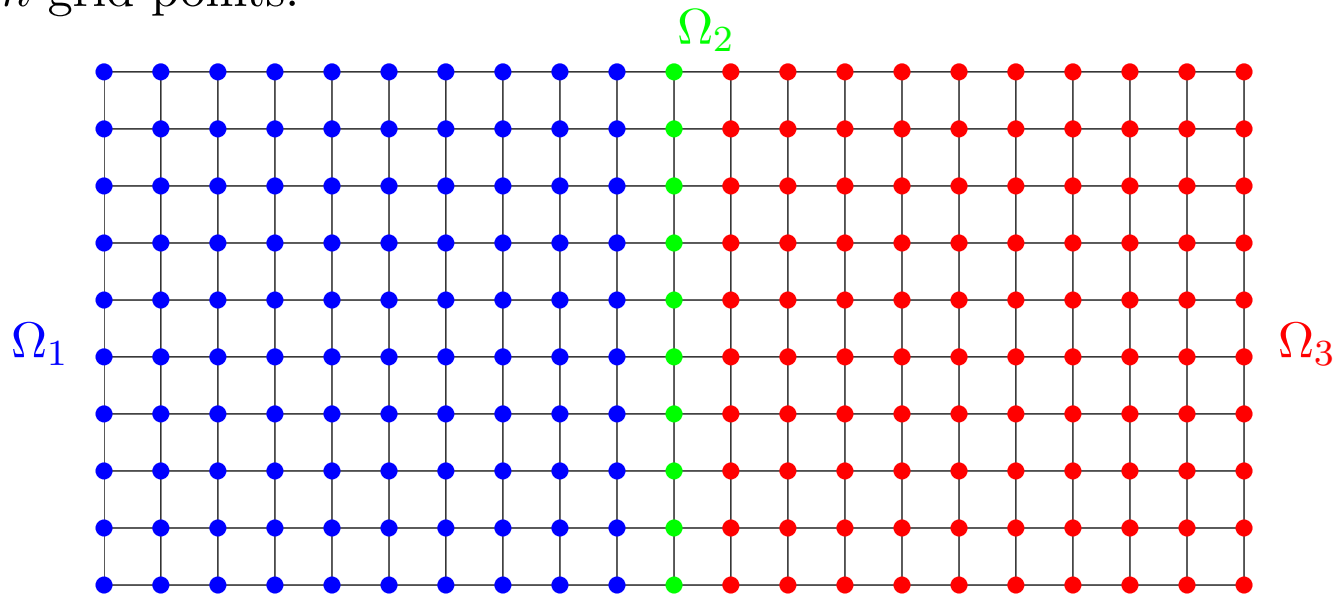
$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

The ranks of $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ are $n = O(N^{0.5})$.

Moreover, $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ consist mostly of zeros in this case!

The "recursion on dimension" method described is very similar to *nested dissection* methods for inverting/factoring sparse matrices arising from the finite element or finite difference discretization of an elliptic PDE.

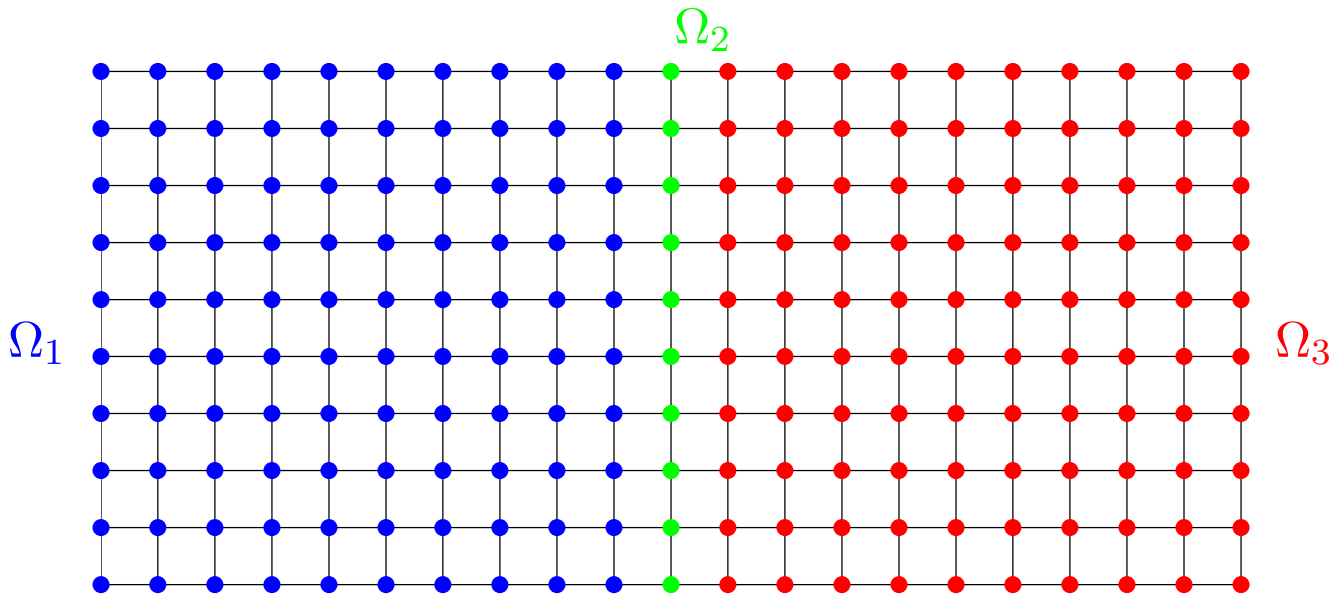
To illustrate, consider a square regular grid for the five-point stencil on a rectangular domain with $N = 2n \times n$ grid points.



The coefficient matrix is tessellated as

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline & \mathbf{A}_{32} & \mathbf{A}_{33} \end{array}\right]$$

Now $\mathbf{A}_{12}$, $\mathbf{A}_{21}^{\mathrm{t}}$, $\mathbf{A}_{23}^{\mathrm{t}}$, and $\mathbf{A}_{32}$ have $n = O(N^{0.5})$ columns.
Having rank $O(N^{0.5})$ is good, but being of *size* $O(N^{0.5})$ is better!

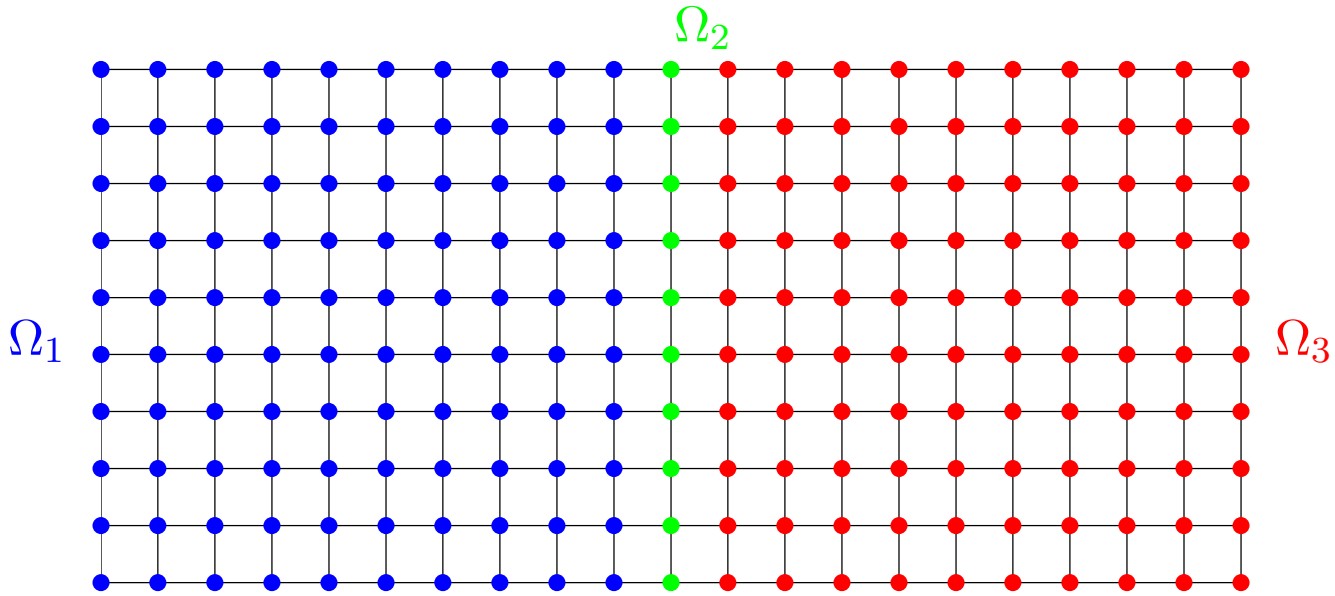


The coefficient matrix is tessellated as

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline & \mathbf{A}_{32} & \mathbf{A}_{33} \end{array}\right]$$

To execute nested dissection, the recursive step consists of 3 tasks:

- Compute the factorization $\mathbf{L}_{11}\,\mathbf{U}_{11} = \mathbf{A}_{11}$.
- Compute the factorization $\mathbf{L}_{33}\,\mathbf{U}_{33} = \mathbf{A}_{33}$.
- Compute the factorization $\mathbf{L}_{22}\,\mathbf{U}_{22} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} - \mathbf{A}_{23}\mathbf{A}_{33}^{-1}\mathbf{A}_{32}$.



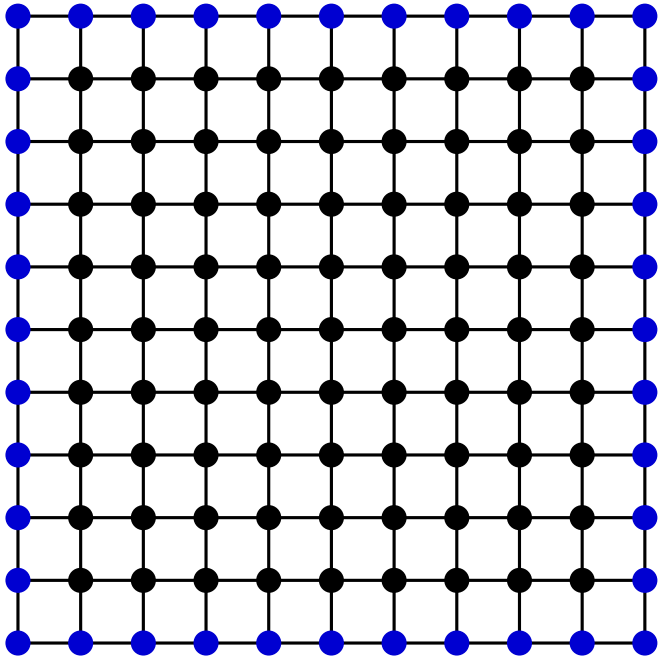
The coefficient matrix is tessellated as

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \\ \hline \mathbf{A}_{12} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \hline & \mathbf{A}_{32} & \mathbf{A}_{33} \end{array}\right]$$

To execute nested dissection, the recursive step consists of 3 tasks:

- Compute the factorization $\mathbf{L}_{11}\,\mathbf{U}_{11} = \mathbf{A}_{11}$.
- Compute the factorization $\mathbf{L}_{33}\,\mathbf{U}_{33} = \mathbf{A}_{33}$.
- Compute the factorization $\mathbf{L}_{22}\,\mathbf{U}_{22} = \underbrace{\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} - \mathbf{A}_{23}\mathbf{A}_{33}^{-1}\mathbf{A}_{32}}_{\text{This is an HSS matrix!}}$.

Example: **Inversion of a "Finite Element Matrix" (with A. Gillman)**



A grid conduction problem — $\mathbf{A}$ is a "five-point stencil" — very large, sparse.

Each link has conductivity drawn from a uniform random distribution on [1, 2].

Solution strategy: Perform nested dissection on the grid. Use HSS algebra to accelerate all computations involving dense matrices larger than a certain threshold. Total complexity is $O(N)$ (as compared to $O(N^{1.5})$ for classical nested dissection).

N	T_{solve} (sec)	T_{apply} (sec)	M (MB)	e_3	e_4
512^2	7.98	0.007	8.4	$2.7523e-6$	$6.6631e-9$
1024^2	26.49	0.014	18.6	-	-
2048^2	98.46	0.020	33.1	-	-
4096^2	435.8	0.039	65.6	-	-

T_{solve} Time required to compute all Schur complements ("set-up time")

T_{apply} Time required to apply a Dirichlet-to-Neumann op. (of size $4\sqrt{N} \times 4\sqrt{N}$)

M Memory required to store the solution operator

e_3 The l^2-error in the vector $\tilde{\mathbf{A}}_{nn}^{-1} r$ where r is a unit vector of random direction.

e_4 The l^2-error in the first column of $\tilde{\mathbf{A}}_{nn}^{-1}$.

Related work:

Solvers of this type have attracted much attention recently, including:

- $\mathcal{H}$-LU factorization of coefficient matrices by L. Grasedyck, S. LeBorne, S.Börm, et al. (2006)

- Multifrontal methods accelerated by HSS-matrix algebra: J. Xia, S. Chandrasekaran, S. Li. (2009)

  Currently large effort at Purdue in this direction. (J. Xia, M. V. de Hoop, et al). Massive computations on seismic wave propagation.

- L. Ying & P. Schmitz — general meshes in 2D, Cartesian meshes in 3D, etc. (2010).

Accelerated nested dissection on grids in $\mathbb{R}^3$

We have implemented the accelerated nested dissection method to solve the electrostatics problems arising in *transcranial magnetic stimulation (TMS)*. This environment is characterized by:

- There is time to preprocess a given geometry.
- Given a load, the solution should be found "instantaneously."

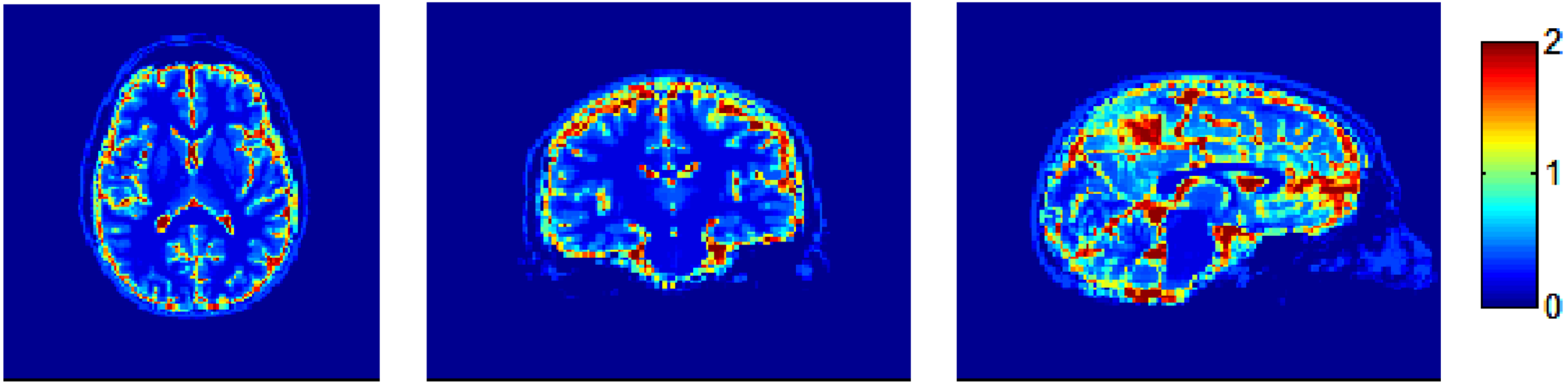


Figure 1: Brain conductivity map. The brain size 77 x 94 x 61. Left to right: Axial, coronal and sagital view.

Joint work with Frantisek Cajko, Luis Gomez, Eric Michielssen, and, Luis Hernandez-Garcia of U. Michigan.

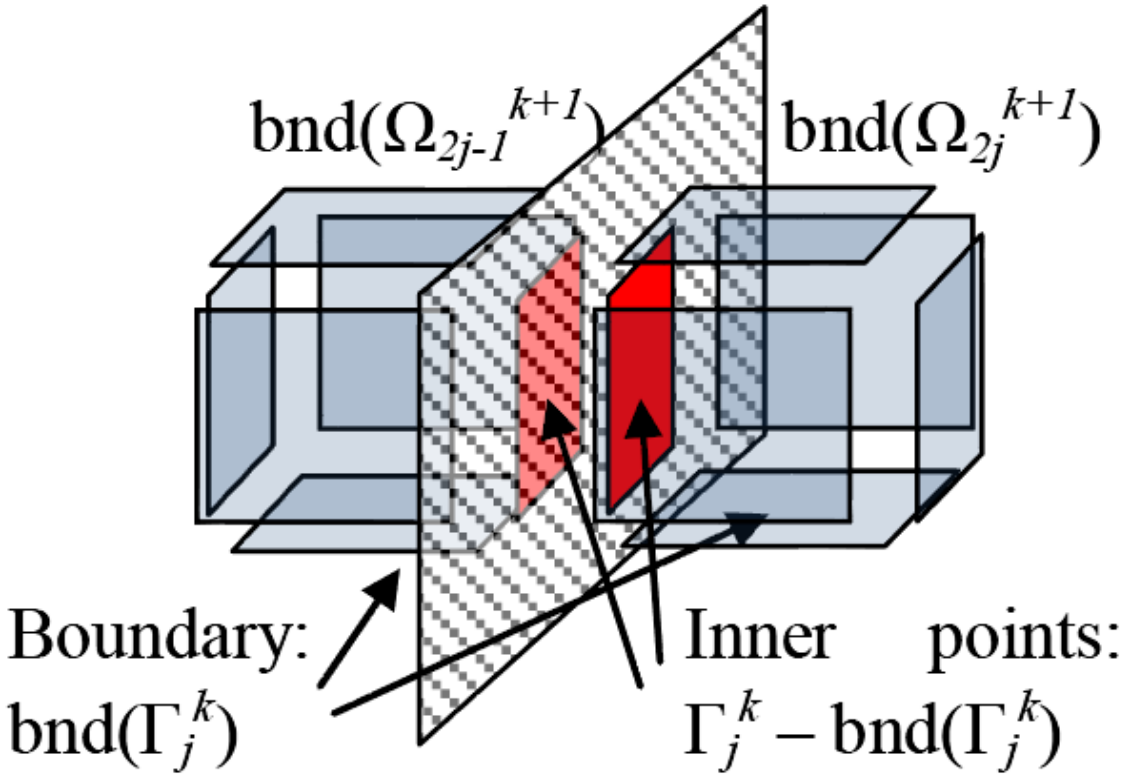


Figure 4: Non-leaf cluster of points and its boundary and inner points. The bisecting plane is shaded.

Numerical results: Single CPU desktop. Local accuracy was 10^{-8}.

Our benchmark was the "Pardiso" package in Intel's MKL.

N	Storage (MB)		Factorization (sec)		Solution (sec)	
	Pardiso	FDS	Pardiso	FDS	Pardiso	FDS
$32^3 = 32768$	157	283	8	53	0.1	0.1
$64^3 = 262144$	2900	3313	549	1683	1.3	1.2
$72^3 = 373248$	4700	4693	1106	3142	2.2	1.9
$81^3 = 531441$	8152	6929	2330	5669	4.1	3.2
$90^3 = 729000$	12500	9550	4630	8658	7.2	4.7

Note: Direct solvers require substantially more memory than, e.g., multigrid.

Note: The gain over Pardiso is modest for small problem sizes, but:

- FDS has better asymptotic scaling as N grows.
- FDS is better suited for large-scale parallelization.
- FDS will be extremely fast for pure "boundary value problems" (this is speculation . . .)
- The FDS methods are still fairly immature; improvements are to be expected.

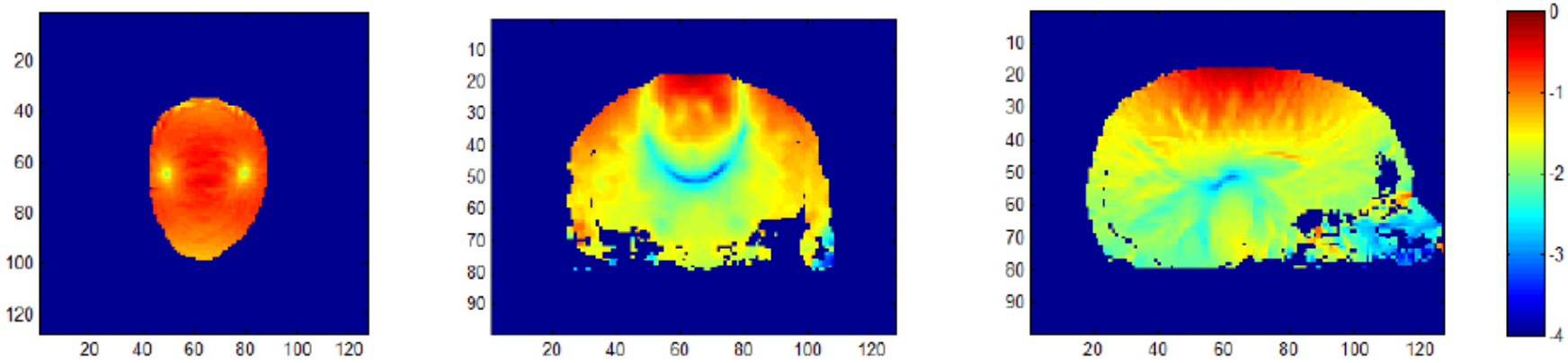


Figure 14: Magnitude of the total electric field inside the head in logarithmic scale. The field is normalized with respect to the highest value in the head. Left to right: Axial (xy-plane, z=24), coronal (xz-palne, y=63) and sagital (yz-plane, x=63) views.

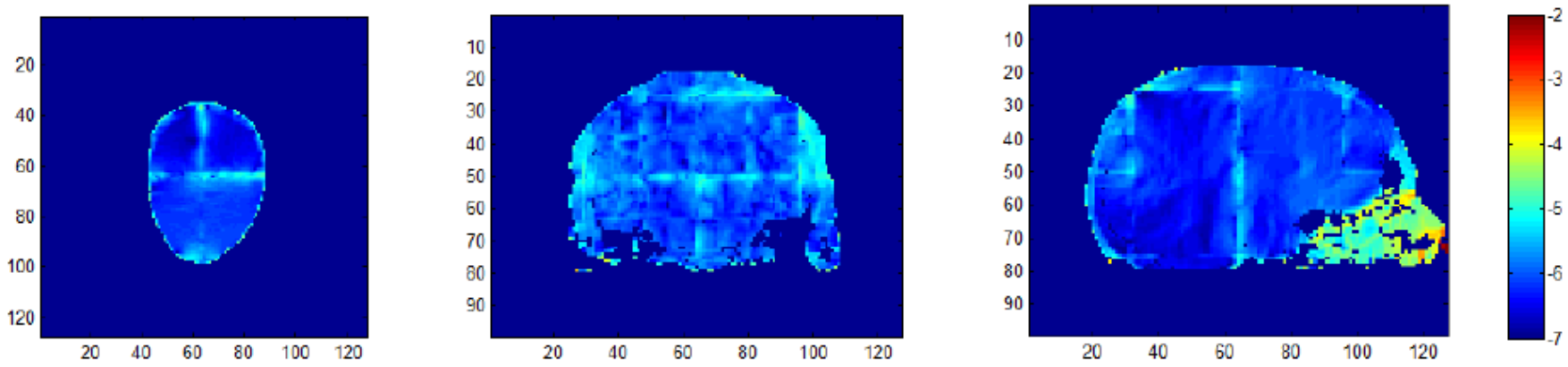


Figure 15: Error magnitude of the total electric field inside the head using a logarithmic scale. The error is the magnitude of the difference between solutions of obtained by the fast direct solver with threshold 1E-8 by the iterative solver with residual 9E-13. The error was normalized to the highest value of the electric field in the head. The largest magnitude of the error is 0.02. Left to right: Axial (z=24), coronal (y=63) and sagital (x=63) views.

There may be short-cuts to finding the inverses ...

Recent work indicates that randomized sampling could be used to very rapidly find a data-sparse representation of a matrix (in $\mathcal{H}$ / $\mathcal{H}^2$ / HSS / ... format).

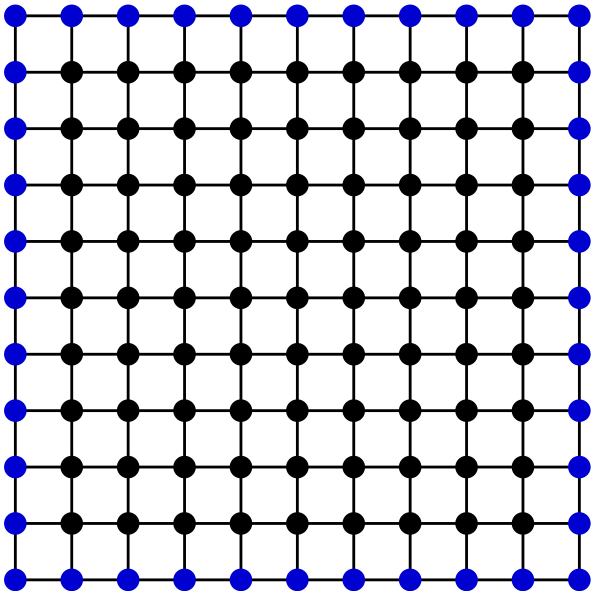
The idea is to extract information by applying the operator to be compressed to a sequence of random vectors. In the present context, "applying the inverse" of course corresponds simply to a linear solve.

- "Fast construction of hierarchical matrix representation from matrix-vector multiplication", L. Lin, J. Lu, L. Ying., *J. of Computational Physics*, **230**(10), 2011.

- P.G. Martinsson, "A fast randomized algorithm for computing a Hierarchically Semi-Separable representation of a matrix". *SIAM J. on Matrix Analysis and Appl.*, **32**(4), pp. 1251–1274, 2011.

For more on randomized sampling in numerical linear algebra, see:
N. Halko, P.G. Martinsson, J. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions."
SIAM Review, **53**(2), 2011. pp. 217–288.

Question: When are the "boundary operators" compressible in HSS form?



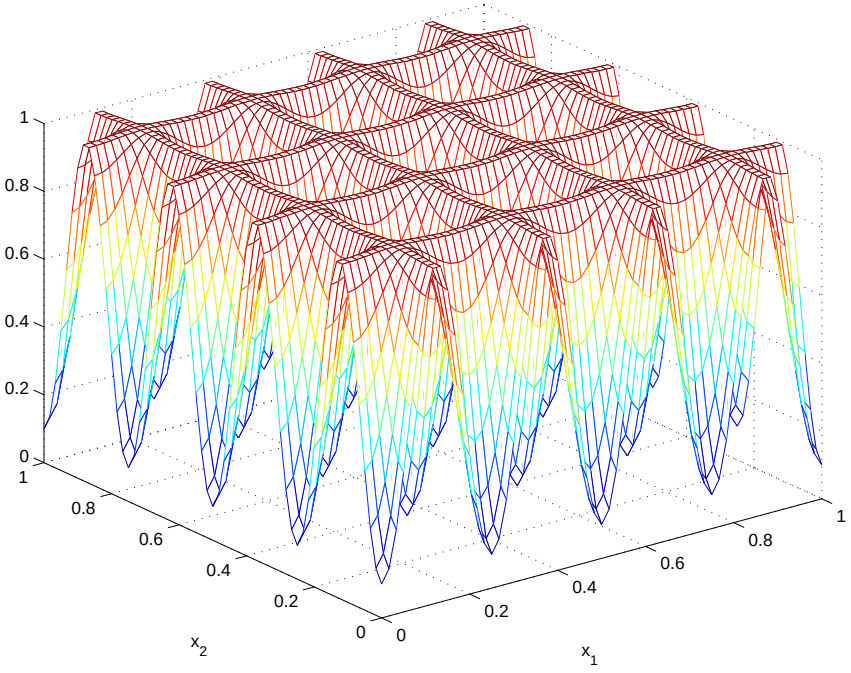
Electrostatics on a network of resistors.

The object computed is the lattice "Neumann-to-Dirichlet" boundary operator. It maps a vector of fluxes on the blue nodes to a vector of potentials on the blue nodes.

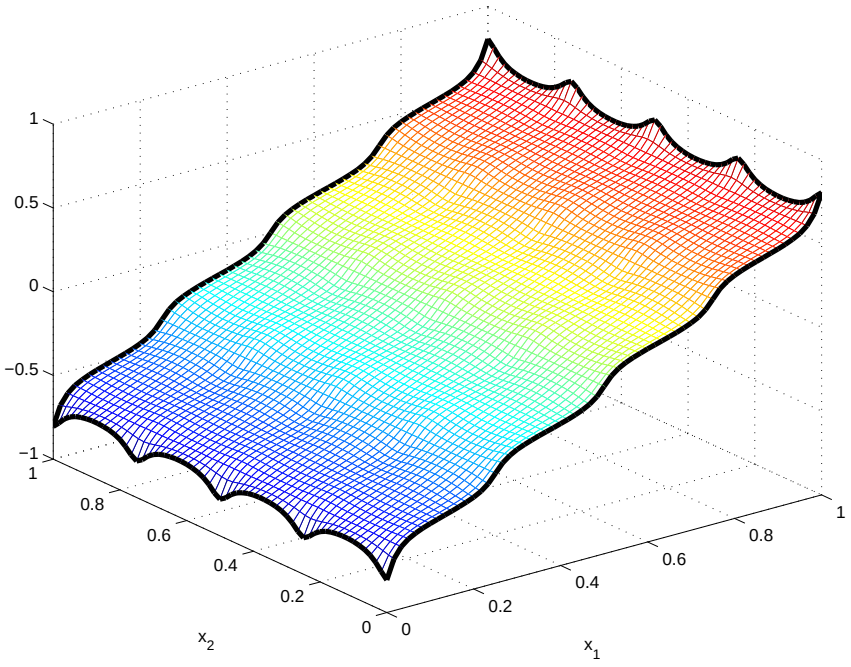
Question: How compressible is the N2D operator?

Case A: Constant conductivities — standard 4/-1/-1/-1/-1 five-point stencil.
Case B: Random conductivities — drawn uniformly from the interval [1, 2].
Case C: Periodic oscillations — high aspect ratio.
Case D: Random cuts — 5% of the bars were cut, the others have unit conductivity.
Case E: Large cracks.

Case C — periodic oscillations:

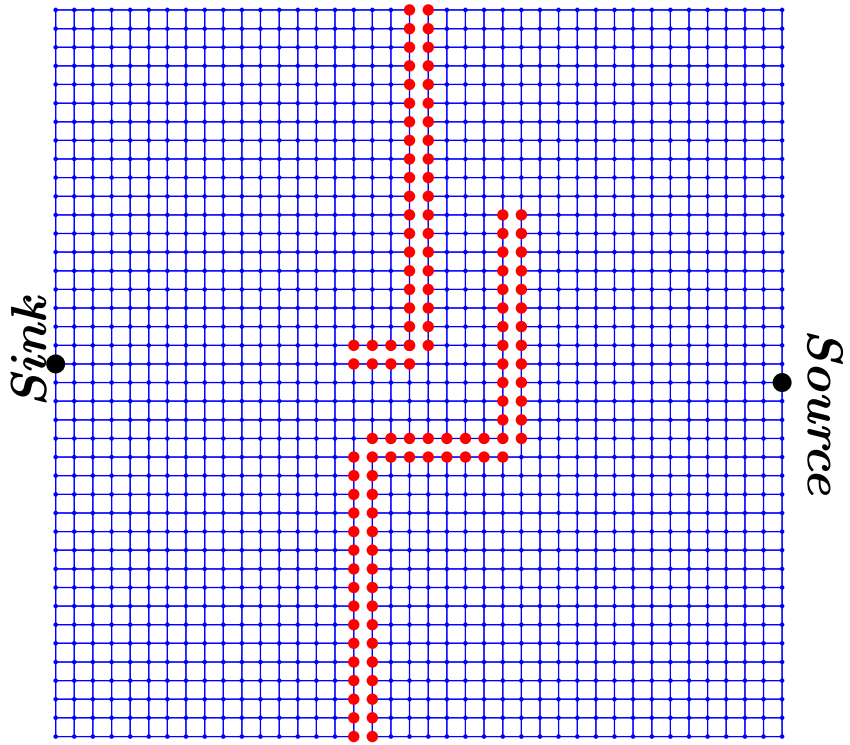


Conductivities

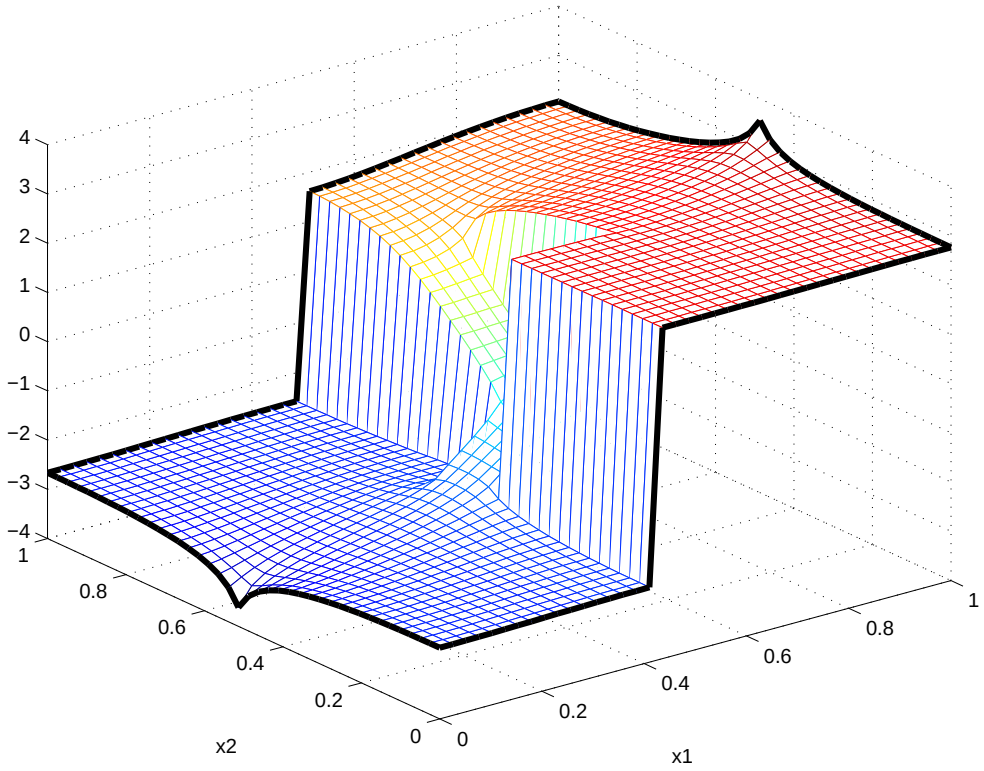


Typical solution field

Case E — large cracks:



Geometry



Permissible solution field

Memory requirements in floats per degree of freedom:

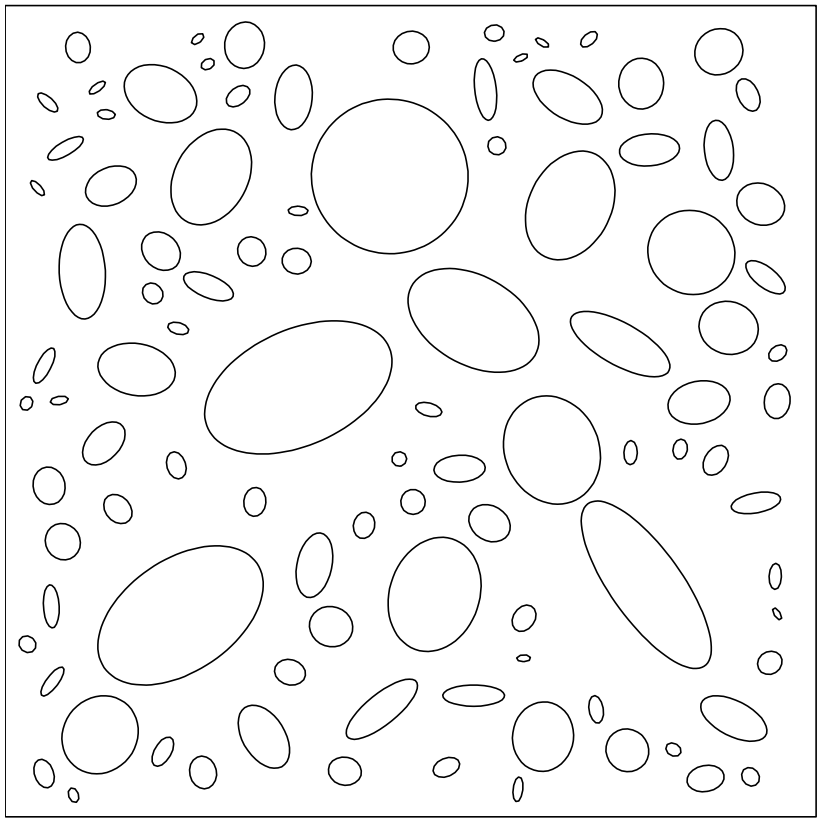
All operators were compressed to a relative accuracy of 10^{-10}.

	$N_{\text{side}} = 100$	$N_{\text{side}} = 200$	$N_{\text{side}} = 400$	$N_{\text{side}} = 800$	$N_{\text{side}} = 1600$
General matrix	396	796	1596	3196	6396
Case A (constant conductivities)	97.7	98.6	98.1	96.8	95.7
Case B (periodic conductivities)	95.9	97.4	96.7	95.4	93.9
Case C (random conductivities)	97.8	99.7	98.8	97.5	96.0
Case D (random cuts)	95.5	97.5	96.6	95.4	94.1
Case E (cracks)	95.7	98.1	97.7	96.8	95.5

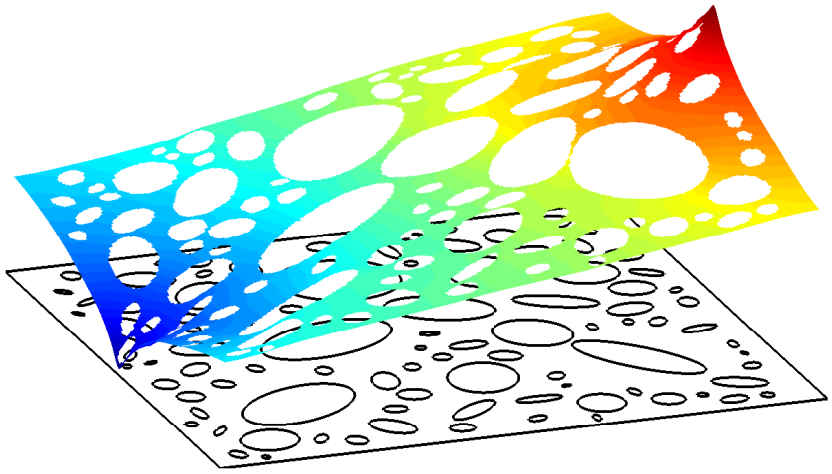
Key observations:

- The amount of memory required is essentially problem independent.
- Almost perfect linear scaling.

A conduction problem on a perforated domain



Geometry



Potential

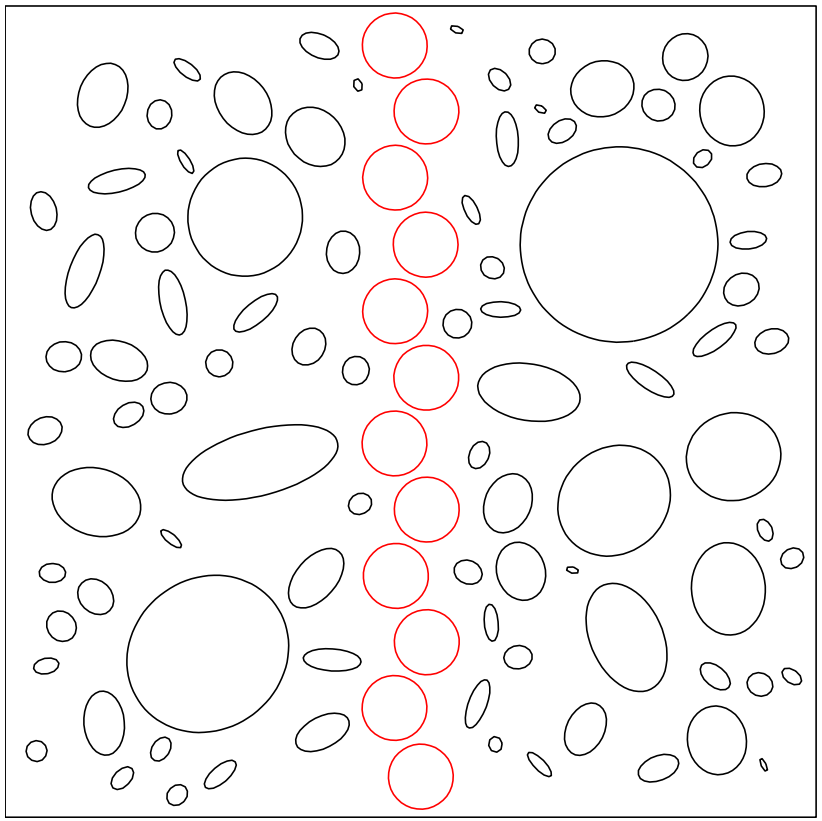
The Neumann-to-Dirichlet operator for the exterior boundary was computed.

The boundary was split into 44 panels, with 26 Gaussian quadrature nodes on each one.

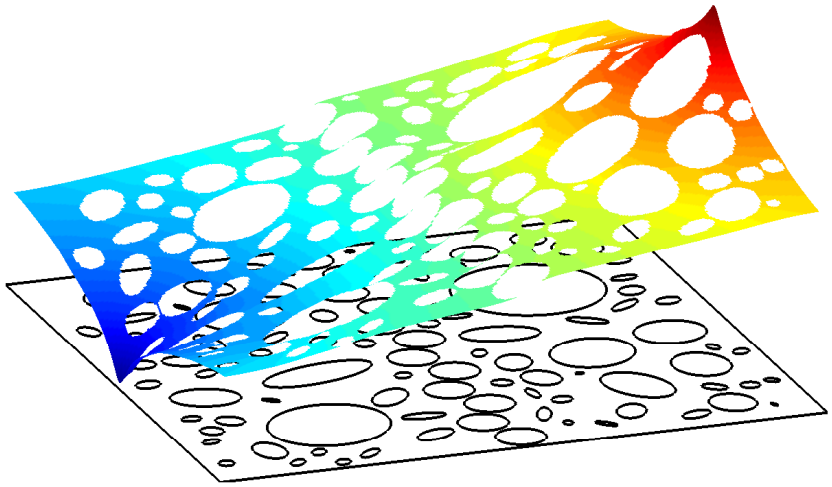
This gives a relative accuracy of 10^{-10} for evaluating fields at points very close to the boundary (up to 0.5% of the side-length removed).

Storing the N2D operator (in a data-sparse format) requires 120 floats per degree of freedom.

A conduction problem on a perforated domain — close to "percolation"



Geometry



Potential

The Neumann-to-Dirichlet operator for the exterior boundary was computed.

The boundary was split into 44 panels, with 26 Gaussian quadrature nodes on each one.

This gives a relative accuracy of 10^{-10} for evaluating fields at points very close to the boundary (up to 0.5% of the side-length removed).

Storing the N2D operator (in a data-sparse format) requires 118 floats per degree of freedom.

Question: When are the "boundary operators" compressible in HSS form?

Apparent answer: Almost always for non-oscillatory problems. (?)

Dense matrices that arise in numerical algorithms for elliptic PDEs are surprisingly well suited to the HSS-representation. The format is robust to:

- Irregular grids.
- PDEs with non-smooth variable coefficients.
- Inversion, LU-factorization, matrix-matrix-multiplies, etc.

For oscillatory problems, the ranks grow as the wave-length of the problem is shrunk relative to the size of the geometry, which eventually renders the direct solvers prohibitively expensive. However, the methodology remains efficient for "surprisingly" small wave-lengths.

Some supporting theory and "intuitive arguments" exist, but the observed performance still exceeds what one would expect, both in terms of the range of applicability and what the actual ranks should be. (At least what I would expect!)

Assertions:

- Fast direct solvers excel for problems on 1D domains. (They should become the default.)
  - Integral operators on the line.
  - Boundary Integral Equations in $\mathbb{R}^2$.
  - Boundary Integral Equations on rotationally symmetric surfaces in $\mathbb{R}^3$.
- Existing fast direct solvers for "finite element matrices" associated with elliptic PDEs in $\mathbb{R}^2$ work very well. In $\mathbb{R}^3$, they can be game-changing in specialized environments.

Predictions:

- For BIEs associated with non-oscillatory problems on surfaces in $\mathbb{R}^3$, the complexity will be reduced from $O(N(\log N)^p)$ to $O(N)$, with a modest scaling constant.
- *Randomized methods* will prove enormously helpful.
  They have already demonstrated their worth in large scale linear algebra.
- Direct solvers for *scattering problems* will find users, even if expensive.
  $O(N^{1.5})$ or $O(N^2)$ flop counts may be OK, provided parallelization is possible.
- Direct solvers will provide a fantastic tool for *numerical homogenization.*

Open questions:

- How efficient can direct solvers be for volume problems in 3D?
- Are $O(N)$ direct solvers for highly oscillatory problems possible?