Fast numerical methods for solving linear PDEs

P.G. Martinsson, The University of Colorado at Boulder

Acknowledgements: Some of the work presented is joint work with Vladimir Rokhlin and Mark Tygert at Yale University.

In this talk, we will discuss numerical methods for solving the equation

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

where Ω is a domain in $\mathbb{R}^2$ or $\mathbb{R}^3$ with boundary Γ.

More generally, we will consider stationary linear Boundary Value Problems

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

such as:

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

Outline of talk:

- *Background:*
  - “Fast” methods and scaling of computational cost.
  - “Iterative” vs. “direct” methods.
  - Existing methodology for “fast” PDE solvers.
- *New: Fast and direct methods for solving PDEs numerically:*
  - Techniques for representing functions and operators.
  - Hierarchical computation of inverse operators.
  - Matrix approximation via randomized sampling.
- *Numerical examples.*

Computational science — background

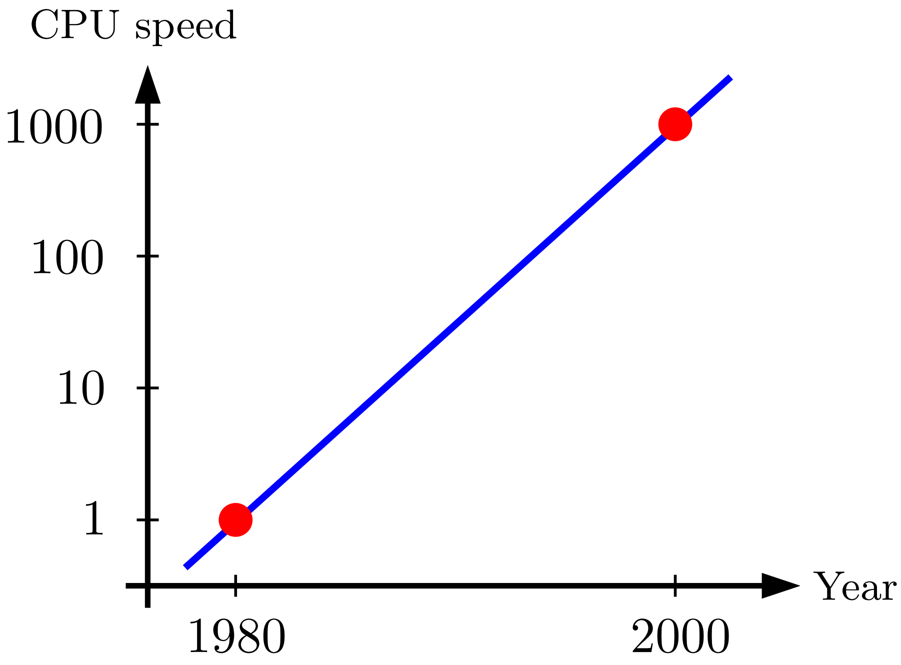
One of the principal developments in science and engineering over the last couple of decades has been the emergence of computational simulations.

We have achieved the ability to computationally model a wide range of phenomena. As a result, complex systems such as cars, micro-chips, new materials, city infra-structures, *etc*, can today be designed more or less entirely in a computer, with little or no need for physical prototyping.

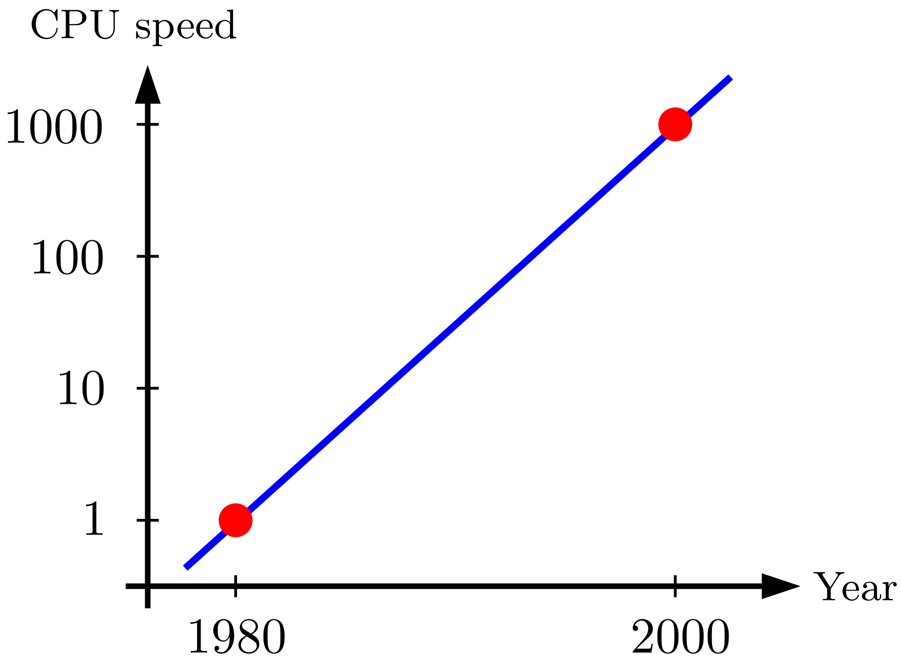
This shift towards computational modelling has in many areas led to dramatic cost savings and improvements in performance.

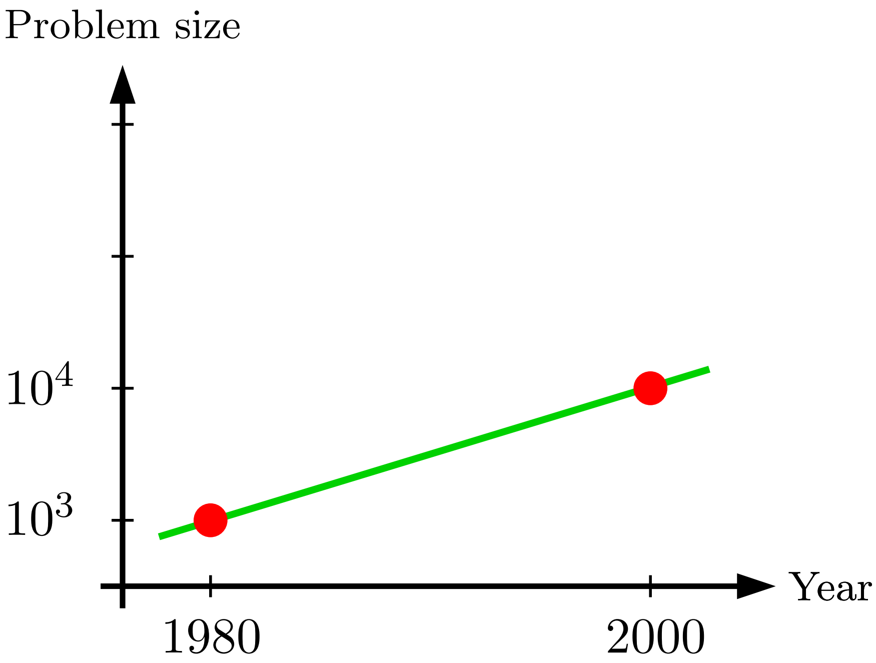
What enabled all this was the development of faster more powerful computers, *and the development of faster algorithms*.

Growth of computing power and the importance of algorithms



Growth of computing power and the importance of algorithms



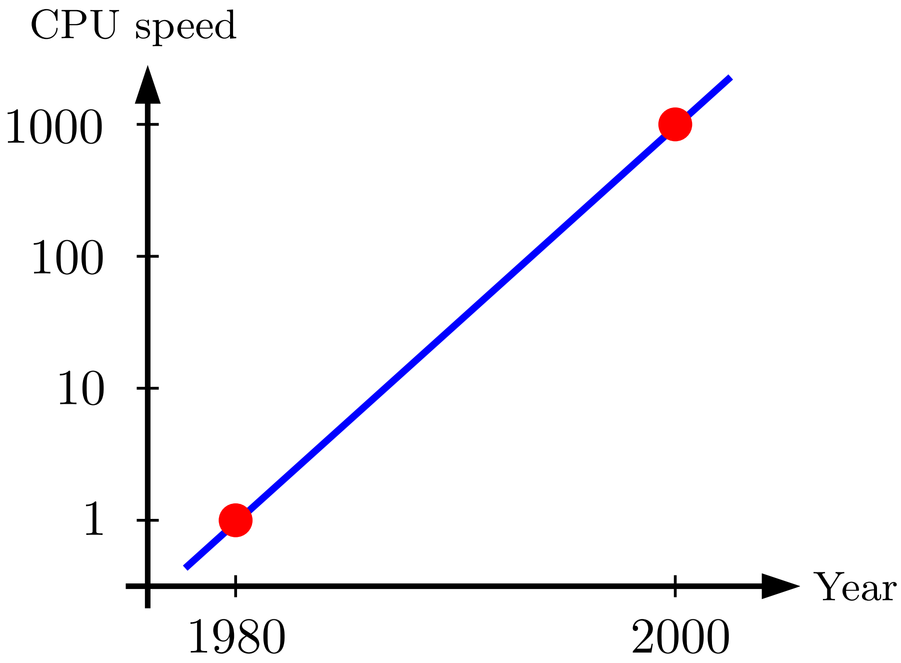


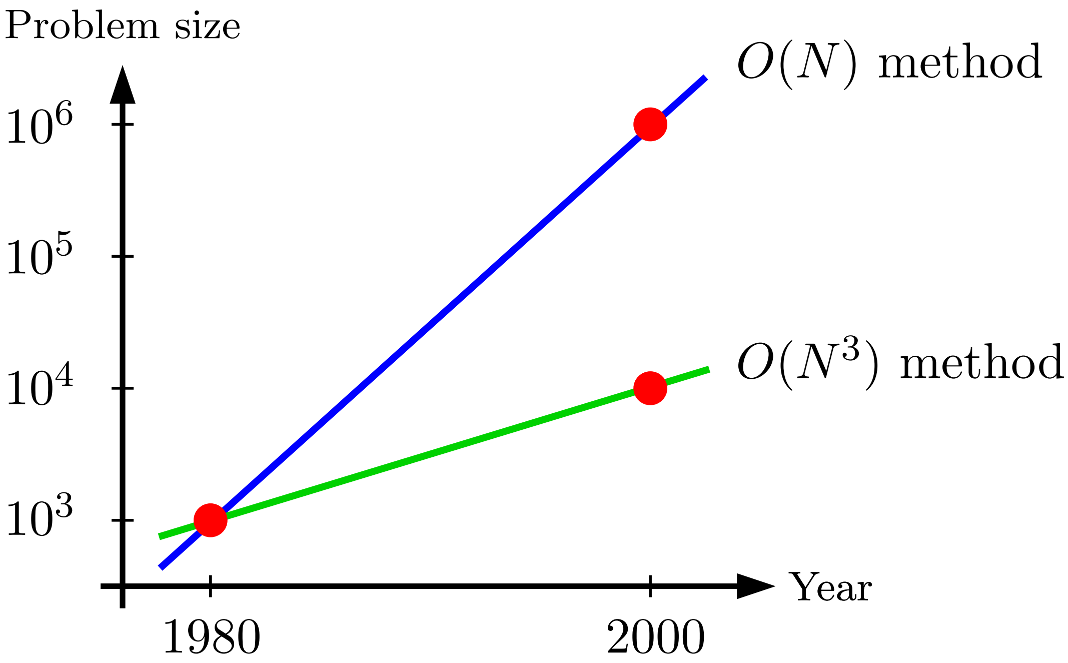
Consider the computational task of solving a linear system of N algebraic equations with N unknowns.

Classical methods such as Gaussian elimination require $O(N^3)$ operations.

Using an $O(N^3)$ method, an increase in computing power by a factor of 1000 enables the solution of problems that are $(1000)^{1/3} = 10$ times larger.

Growth of computing power and the importance of algorithms





Consider the computational task of solving a linear system of N algebraic equations with N unknowns.

Classical methods such as Gaussian elimination require $O(N^3)$ operations.

Using an $O(N^3)$ method, an increase in computing power by a factor of 1000 enables the solution of problems that are $(1000)^{1/3} = 10$ times larger.

Using a method that scales as $O(N)$, problems that are 1000 times larger can be solved.

Definition of the term “fast”:

We say that a numerical method is “fast” if its computational speed scales as $O(N)$ as the problem size N grows.

Methods whose complexity is $O(N\log(N))$ or $O(N(\log N)^2)$ are also called “fast”.

Caveat: It appears that Moore's law is no longer operative.

Processor speed is currently increasing quite slowly.

The principal increase in computing power is coming from parallelization.

Successful algorithms must scale well both with problem size and with the number of processors that a computer has.

The methods of this talk all parallelize naturally.

“Iterative” versus ”direct” solvers

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

$$A\,x = b.$$

Direct methods:

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

Directly access elements or blocks of A.

Always give an answer. Robust.

Deterministic.No convergence analysis.

Have often been considered too slow for high performance computing.

Iterative methods:

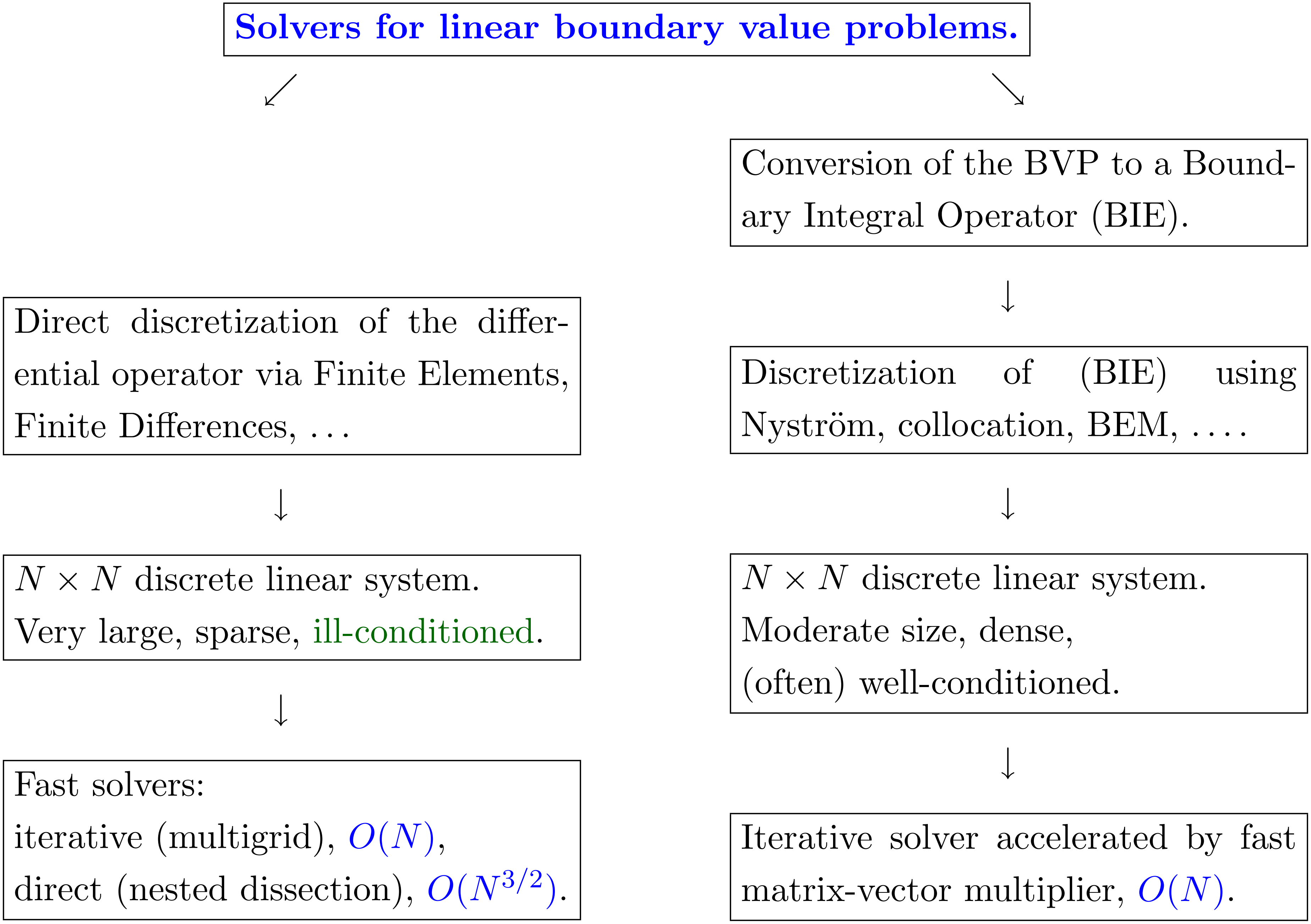
Examples: GMRES, conjugate gradients, Gauss-Seidel, *etc.*

Construct a sequence of vectors $x_1,\,x_2,\,x_3,\,\dots$ that (hopefully!) converge to the exact solution.

Many iterative methods access A only via its action on vectors.

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

Often require problem specific preconditioners.



Solvers for 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.

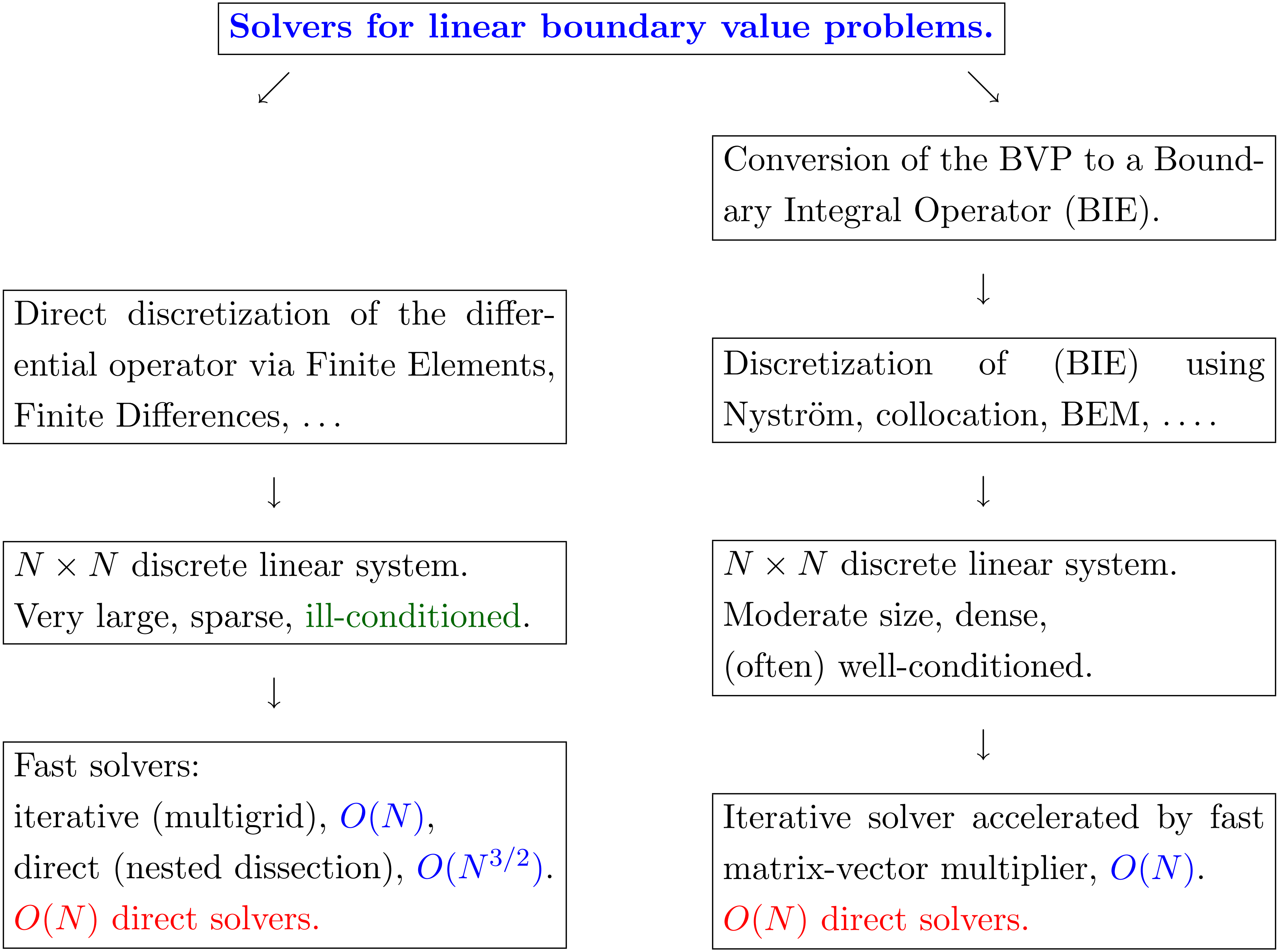
Fast solvers:
iterative (multigrid), $O(N)$,
direct (nested dissection), $O(N^{3/2})$.

Conversion of the BVP to a Boundary Integral Operator (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, $O(N)$.



Solvers for 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 Operator (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.

Reformulating a BVP as a Boundary Integral Equation.

The idea is to convert a linear partial differential equation

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

to an "equivalent" integral equation

$$\text{(BIE)}\qquad v(x) + \int_\Gamma k(x,y)\,v(y)\,ds(y) = h(x), \qquad x \in \Gamma.$$

Example:

Let us consider the equation

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

We make the following Ansatz:

$$u(x) = \int_\Gamma \big(n(y) \cdot \nabla_y \log|x-y|\big) v(y)\, ds(y), \qquad x \in \Omega,$$

where $n(y)$ is the outward pointing unit normal of Γ at y. Then the boundary charge distribution v satisfies the Boundary Integral Equation

$$\text{(BIE)} \qquad v(x) + 2\int_\Gamma \big(n(y) \cdot \nabla_y \log|x-y|\big) v(y)\, ds(y) = 2f(x), \qquad x \in \Gamma.$$

- (BIE) and (BVP) are in a strong sense equivalent.
- (BIE) is appealing mathematically (2nd kind Fredholm equation).

The BIE formulation has powerful arguments in its favor (reduced dimension, well-conditioned, *etc*) that we will return to, but it also has a major drawback:

Discretization of integral operators typically results in dense matrices.

In the 1950's when computers made numerical PDE solvers possible, researchers faced a grim choice:

PDE-based:	Ill-conditioned, N is too large, low accuracy.
Integral Equations:	Dense system.

In most environments, the integral equation approach turned out to be simply too expensive.

(A notable exception concerns methods for dealing with scattering problems.)

The situation changed dramatically in the 1980's. It was discovered that while K_N (the discretized integral operator) is dense, it is possible to evaluate the matrix-vector product

$$v \mapsto K_N\, v$$

in $O(N)$ operations — to high accuracy and with a small constant.

A very succesful such algorithm is the Fast Multipole Method by Rokhlin and Greengard (circa 1985).

Combining such methods with iterative solvers (GMRES / conjugate gradient / . . .) leads to very fast solvers for the integral equations, especially when second kind Fredholm formulations are used.

A prescription for rapidly solving BVPs:

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

Convert (BVP) to a second kind Fredholm equation:

(BIE)
$$u(x) + \int_\Gamma \bigl(n(y)\cdot\nabla_y \log|x-y|\bigr)u(y)\,ds(y) = f(x), \qquad x \in \Gamma.$$

Discretize (BIE) into the discrete equation

(DISC)
$$(I + K_N)\,u_N = f_N$$

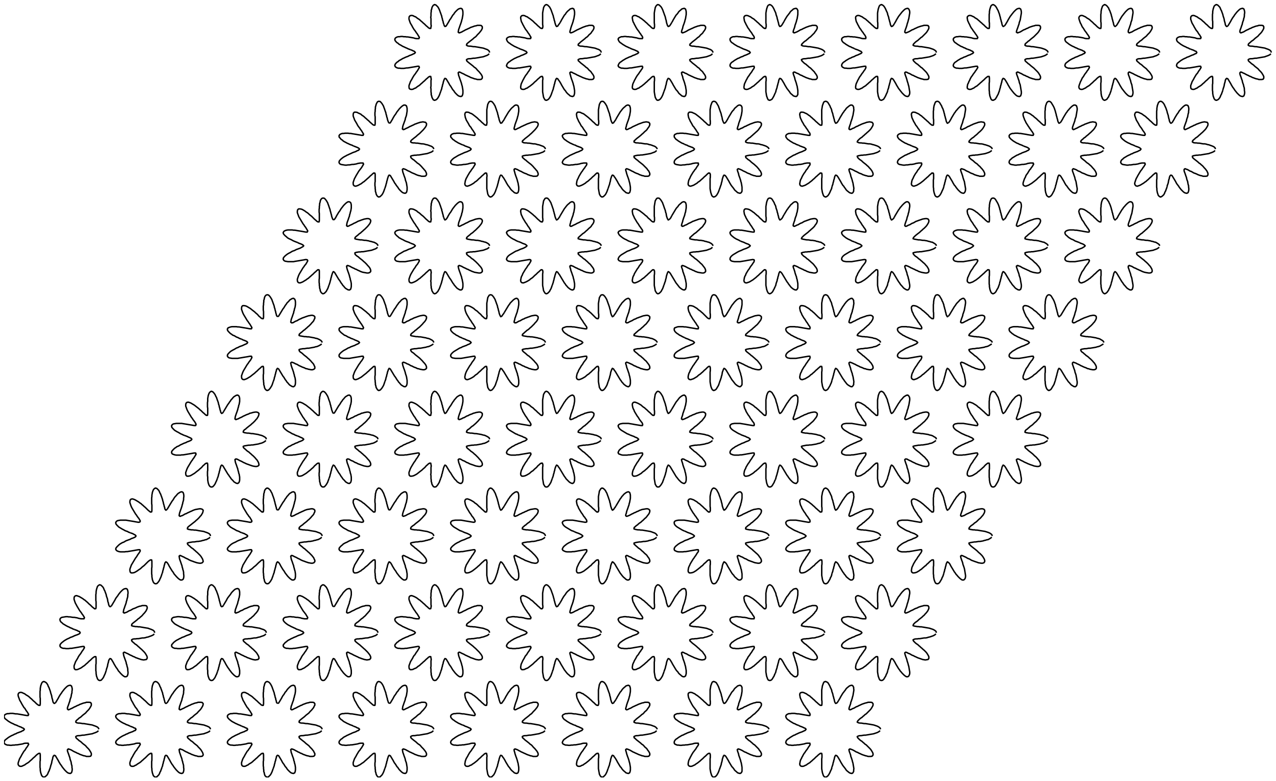
where K_N is a (typically dense) $N \times N$ matrix.

Fast Multipole Method — Can multiply K_N by a vector in $O(N)$ time.

Iterative solver — Solves (DISC) using $\sqrt{\kappa}$ matrix-vector multiplies, where κ is the condition number of $(I + K_N)$.

Total complexity — $O(\sqrt{\kappa}\,N)$. (Recall that κ is small. Like 14.)

Example:



External Laplace problem with Dirichlet boundary data.

The contour is discretized into 25 600 points.

A single matrix-vector multiply takes 0.2 sec on a 2.8 Ghz desktop PC.

Fifteen iterations required for 10^{-10} accuracy → total CPU time is 3 sec.

BIE formulations exist for many classical BVPs

Laplace	$-\Delta u = f,$	
Elasticity	$\frac{1}{2}E_{ijkl}\left(\frac{\partial^2 u_k}{\partial x_l \partial x_j} + \frac{\partial^2 u_l}{\partial x_k \partial x_j}\right) = f_i,$	
Stokes	$\Delta \mathbf{u} = \nabla p, \qquad \nabla \cdot \mathbf{u} = 0,$	
Heat equation	$-\Delta u = -u_t$	(On the surface of $\Omega \times [0,\, T]$.)
Helmholtz	$(-\Delta - k^2)u = f,$	
Schrödinger	$(-\Delta + V)\,\Psi = i\,\Psi_t$	(In the frequency domain.)
Maxwell	$\begin{cases} \nabla \cdot \mathbf{E} = \rho & \nabla \times \mathbf{E} = \ -\frac{\partial \mathbf{B}}{\partial t} \\ \nabla \cdot \mathbf{B} = 0 & \nabla \times \mathbf{B} = \mathbf{J} + \frac{\partial \mathbf{E}}{\partial t} \end{cases}$	(In the frequency domain.)

We have described two paradigms for numerically solving BVPs:

PDE formulation ⇔ Integral Equation formulation

Which one should you choose?

When it is applicable, compelling arguments favor the use of the IE formulation:

Dimensionality:

Frequently, an IE can be defined on the boundary of the domain.

Integral operators are benign objects:

It is (relatively) easy to implement high order discretizations of integral operators. Relative accuracy of 10^{-10} or better is often achieved.

Conditioning:

When there exists an IE formulation that is a Fredholm equation of the second kind, the mathematical equation itself is well-conditioned.

However, integral equation based methods are quite often not a choice:

Fundamental limitations: They require the existence of a fundamental solution to the (dominant part of the) partial difference operator. In practise, this means that the (dominant part of the) operator must be linear and constant-coefficient.

Practical limitations: The infra-structure for BIE methods is underdeveloped. Engineering strength code does not exist for many problems that are very well suited for BIE formulations. The following major pieces are missing:

- **Generic techniques for reformulating a PDE as an integral equation.** We do know how to handle "standard environments", however.
- **Machinery for representing surfaces. Quadrature formulas.** The dearth of tools here has seriously impeded progress on 3D problems.
- **Fast solvers need to be made more accessible and more robust.** Towards this end, we are currently developing *direct solvers* to replace and complement existing iterative ones.

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.)

2. Problems that are relatively ill-conditioned:
   - Scattering problems at intermediate or high 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.

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 should be viewed as a step towards getting a LAPACK-type environment for solving the basic linear boundary value problems of mathematical physics.

Sampling of related work:

1991 Sparse matrix algebra / wavelets, *Beylkin, Coifman, Rokhlin,*

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, et al,*

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

2002 inversion of "Hierarchically semi-separable" matrices, *M. Gu, S. Chandrasekharan, et al.*

2007 factorization of discrete Laplace operators, *S. Chandrasekharan, M. Gu, X.S. Li, J. Xia.*

Fast direct methods — technical issues:

The fast direct methods have many similarities with existing fast schemes such as the Fast Multipole Method. They are all multi-level schemes relying on hierarchical tessellations of the computational domain.

However, a number of technical innovations have been made to achieve high performance. We will discuss the following three:

1. Techniques for representing potentials and operators.

2. Fast inversion schemes.

3. Randomized sampling techniques for approximating matrices.

Item 1 (out of 3): Representation of operators

The fast direct schemes described in this talk rely crucially on rank deficiencies in the off-diagonal blocks of the relevant operators.

This makes the techniques inherently restricted to equations such as:

- Laplace's equation.
- The equations of elasticity.
- Stokes' equation.
- Helmholtz equation at low frequencies.
- Yukawa's equation.

We cannot currently handle:

- High-frequency Helmholtz.
- High-frequency Maxwell.

The problems that we can handle are characterized by "smoothing operators" and equations that exhibit "information loss".

The concepts of “smoothing operators” and “information loss” are closely related to the Saint Venant’s principle in mechanics.

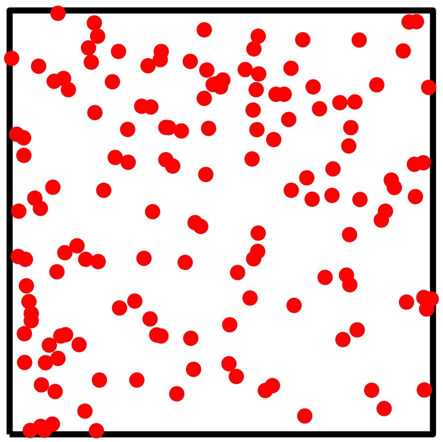
As an example, consider an operator A that maps a vector of electric charges q in a (source) box $\Omega_{\rm s}$ to a vector of potentials v in a (target) box $\Omega_{\rm t}$,

$$v = A\,q.$$

Using complex arithmetic, the entries of A are

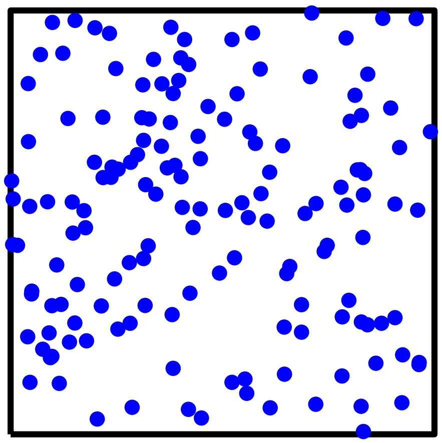
$$A_{mn} = \log(x_m - y_n).$$

The matrix A is *dense*, so the cost of calculating $A\,q$ is $O(M\,N)$.



N source points y_n in $\Omega_{\rm s}$.

Given charges $\{q_n\}_{n=1}^{N}$.



M target points x_m in $\Omega_{\rm t}$.

Sought potentials $\{v_m\}_{m=1}^{M}$.

Multipole expansion: To precision ε, we wish to evaluate

$$v_m = \sum_{n=1}^{N} \log(x_m - y_n)\, q_n, \qquad \text{for } m = 1, \dots, M. \tag{9}$$

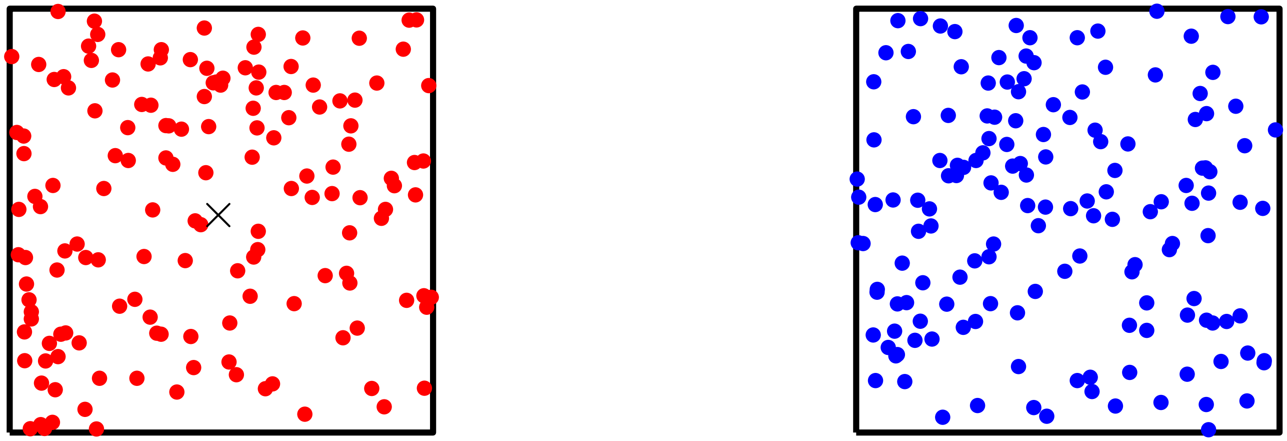
If $|x_m| \geq R_1$ and $|y_n| \leq R_2$, then

$$\log(x_m - y_n) = \log(x_m) - \sum_{j=1}^{k} \frac{y_n^j}{j\, x_m^j} + O\left(\left(\frac{R_2}{R_1}\right)^{k+1}\right). \tag{10}$$

Combining (9) and (10) we obtain

$$v_m = \underbrace{\left(\sum_{n=1}^{N} q_n\right)}_{=\alpha_0} \log(x_m) + \sum_{j=1}^{k} \underbrace{\left(\sum_{n=1}^{N} -\frac{y_n^j}{n}\, q_n\right)}_{=\alpha_j} \frac{1}{x_m^j} + \varepsilon.$$

- Step 1: Compute $\{\alpha_j\}_{j=0}^{k}$ from $\{q_n\}_{n=1}^{N}$. Cost $\sim k\, N$.
- Step 2: Compute $\{v_m\}_{m=1}^{M}$ from $\{\alpha_j\}_{j=0}^{k}$. Cost $\sim k\, M$.

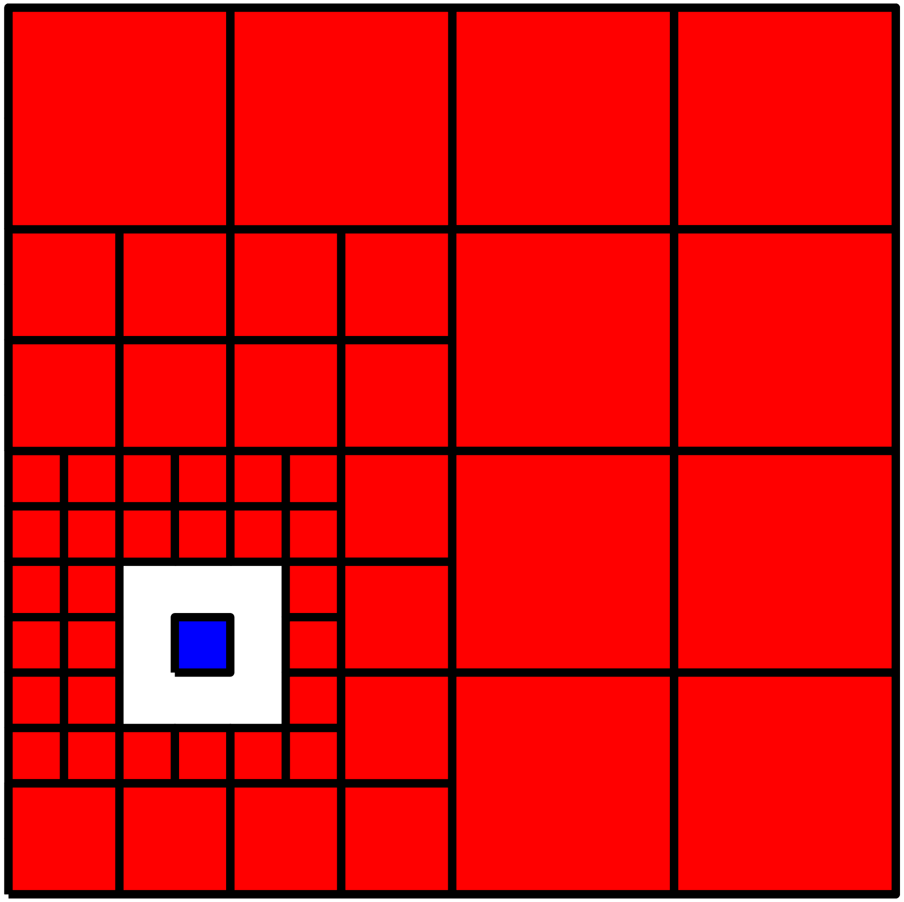


$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[r, "A"] \arrow[d] & \{v_m\}_{m=1}^{M} \\
\{\alpha_j\}_{j=0}^{k} \arrow[ur] &
\end{tikzcd}$$

We have reduced the computational cost from $O(M\,N)$ to $O(k\,(M+N))$.

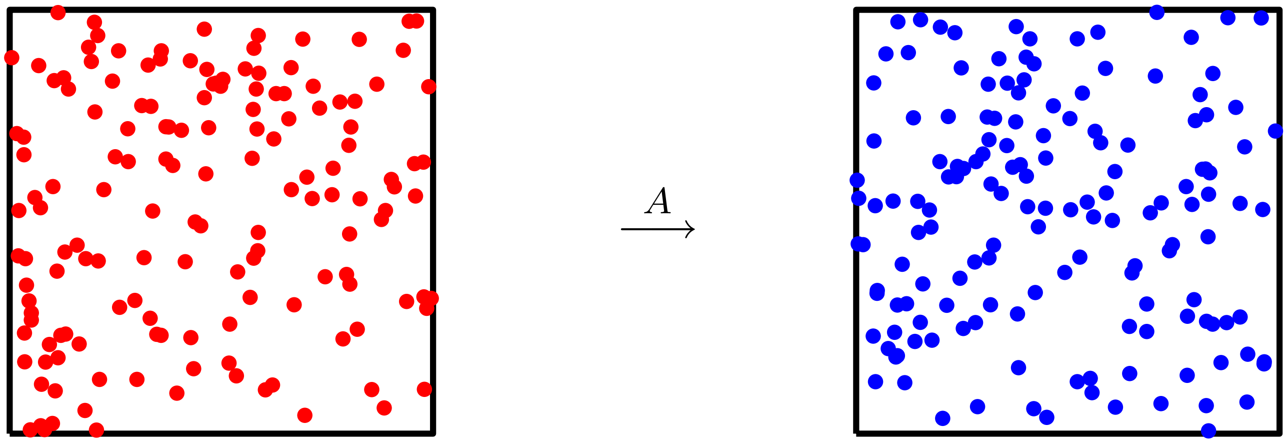
The requested accuracy ε satisfies $\varepsilon \sim \left(\frac{R/\sqrt{2}}{3R/2}\right)^{k+1}$, so $k \sim \frac{\log|\varepsilon|}{\log(3/\sqrt{2})}$.

When the target region and the source regions are the same, we tessellate space into a hierarchy of boxes.



A naïve implementation (the "Barnes-Hut" scheme) leads to: Cost $\sim |\log\varepsilon|\,(\log\ N)\,N$

With some refinements (the "Fast Multipole Method") we obtain: Cost $\sim |\log\varepsilon|^2\,N$.



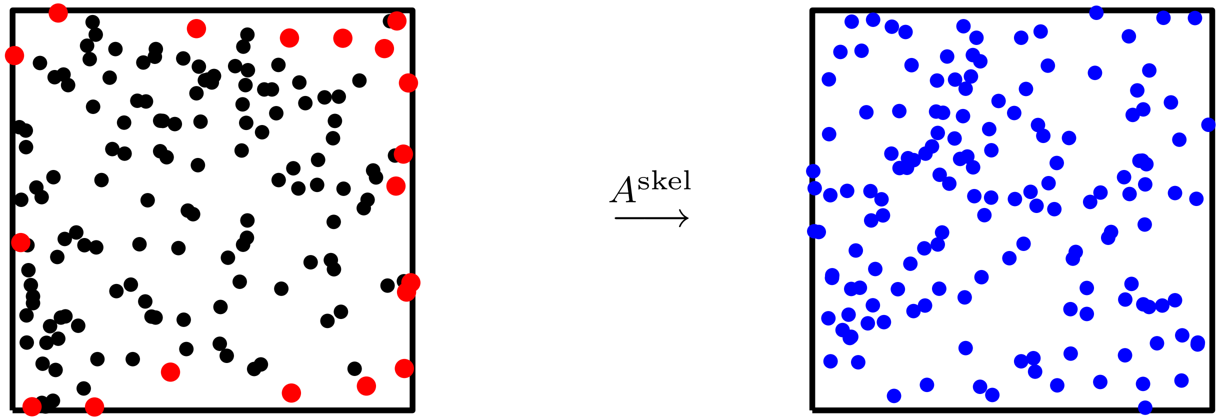
Sources $\{q_n\}_{n=1}^{N}$

Potentials $\{v_m\}_{m=1}^{M}$

$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[rr, "A"] \arrow[d, "?"'] & & \{v_m\}_{m=1}^{M} \\
\text{“Small” representation} \arrow[urr, "?"'] & &
\end{tikzcd}$$

The key observation is that $k = \text{rank}(A) < \min(M, N)$.

Skeletonization

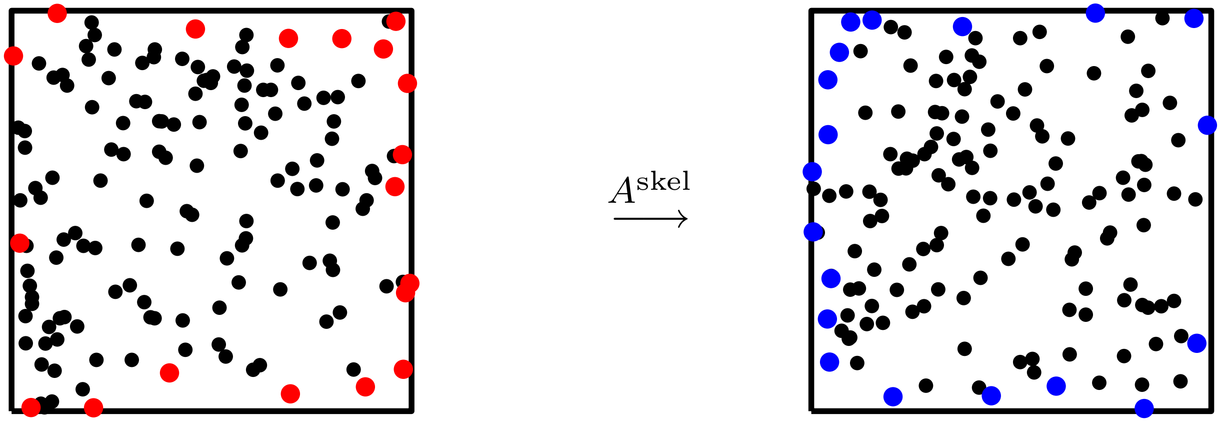


$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[r, "A"] \arrow[d, "U_{\mathrm{s}}^{\mathrm{t}}"'] & \{v_m\}_{m=1}^{M} \\
\{\tilde{q}_{n_j}\}_{j=1}^{k} \arrow[ur, "A^{\mathrm{skel}}"'] &
\end{tikzcd}$$

We can pick k points in Ω_{s} with the property that any potential in Ω_{t} can be replicated by placing charges on these k points.

- The choice of points does not depend on $\{q_n\}_{n=1}^{N}$.
- A^{skel} is a submatrix of A.
- U_{s} contains a $k \times k$ identity matrix.

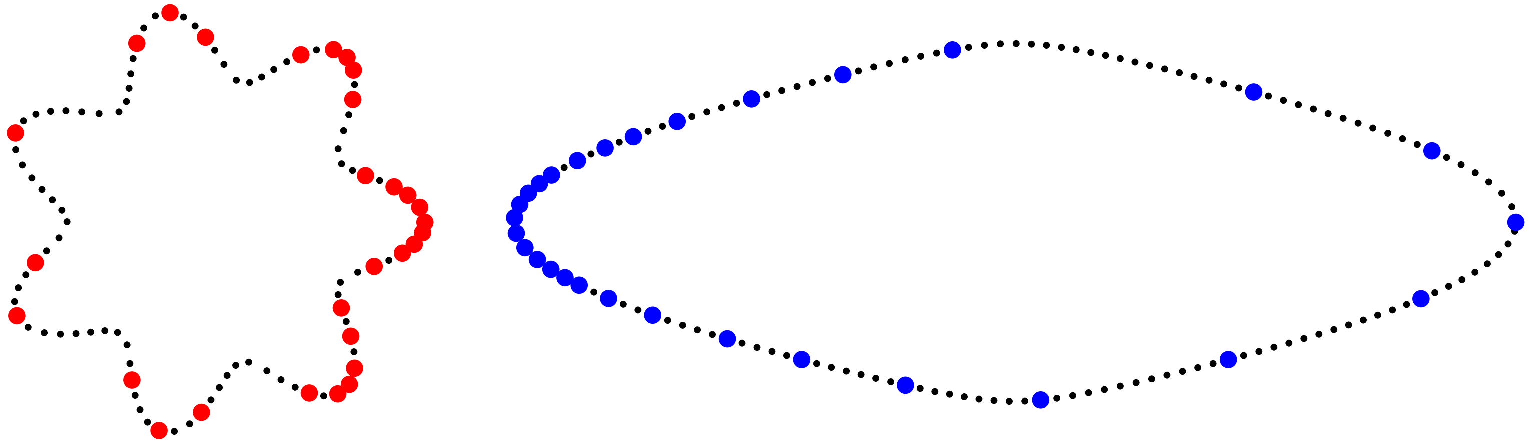
We can "skeletonize" both Ω_{s} and Ω_{t}.



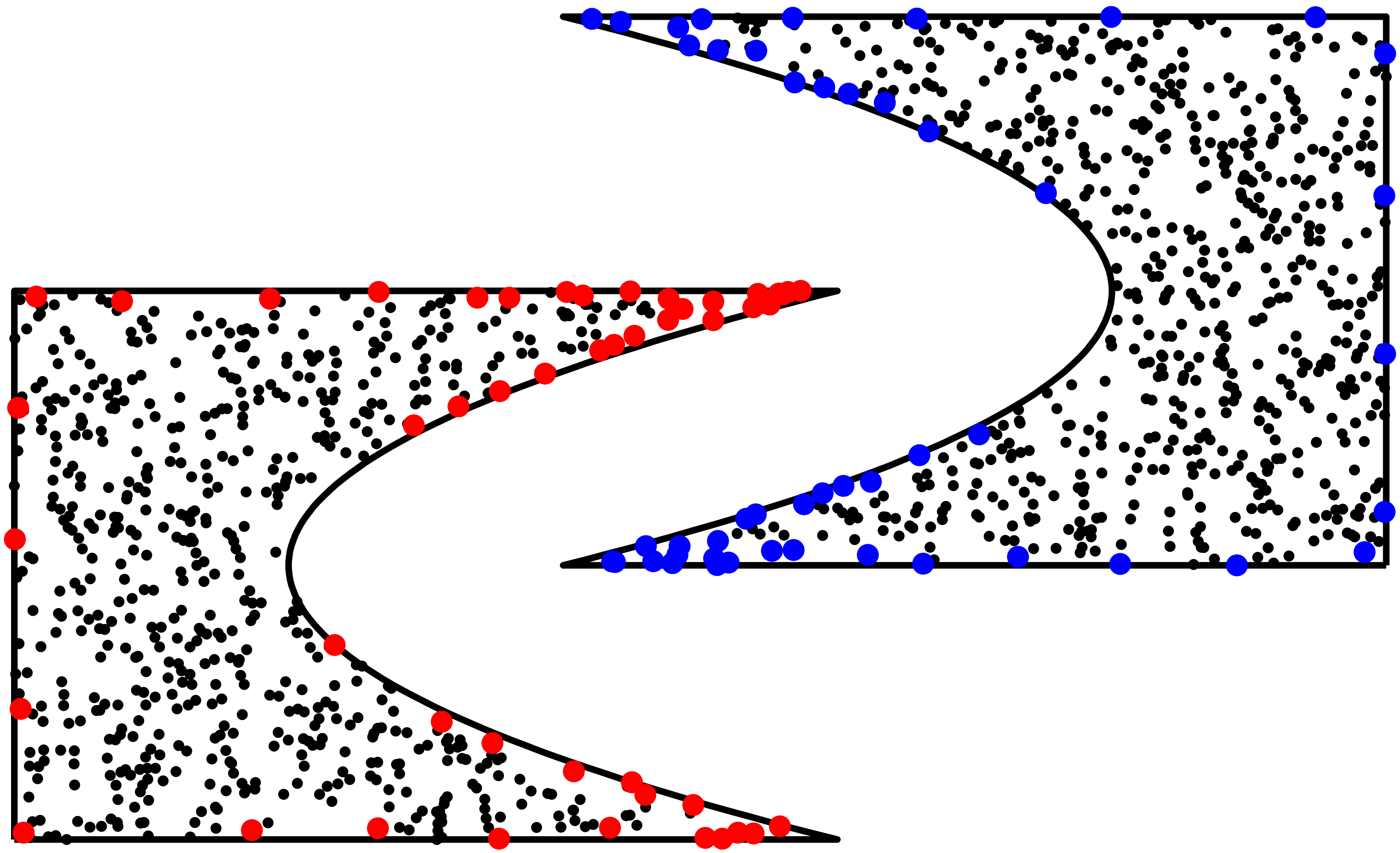
$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[r, "A"] \arrow[d, "U_{\text{s}}^{\text{t}}"'] & \{v_m\}_{m=1}^{M} \\
\{\tilde{q}_{n_j}\}_{j=1}^{k} \arrow[r, "A^{\text{skel}}"'] & \{v_{m_j}\}_{j=1}^{k} \arrow[u, "U_{\text{t}}"']
\end{tikzcd}$$

Rank $= 19$ at $\varepsilon = 10^{-10}$.

Skeletonization can be performed for Ω_{s} and Ω_{t} of various shapes.

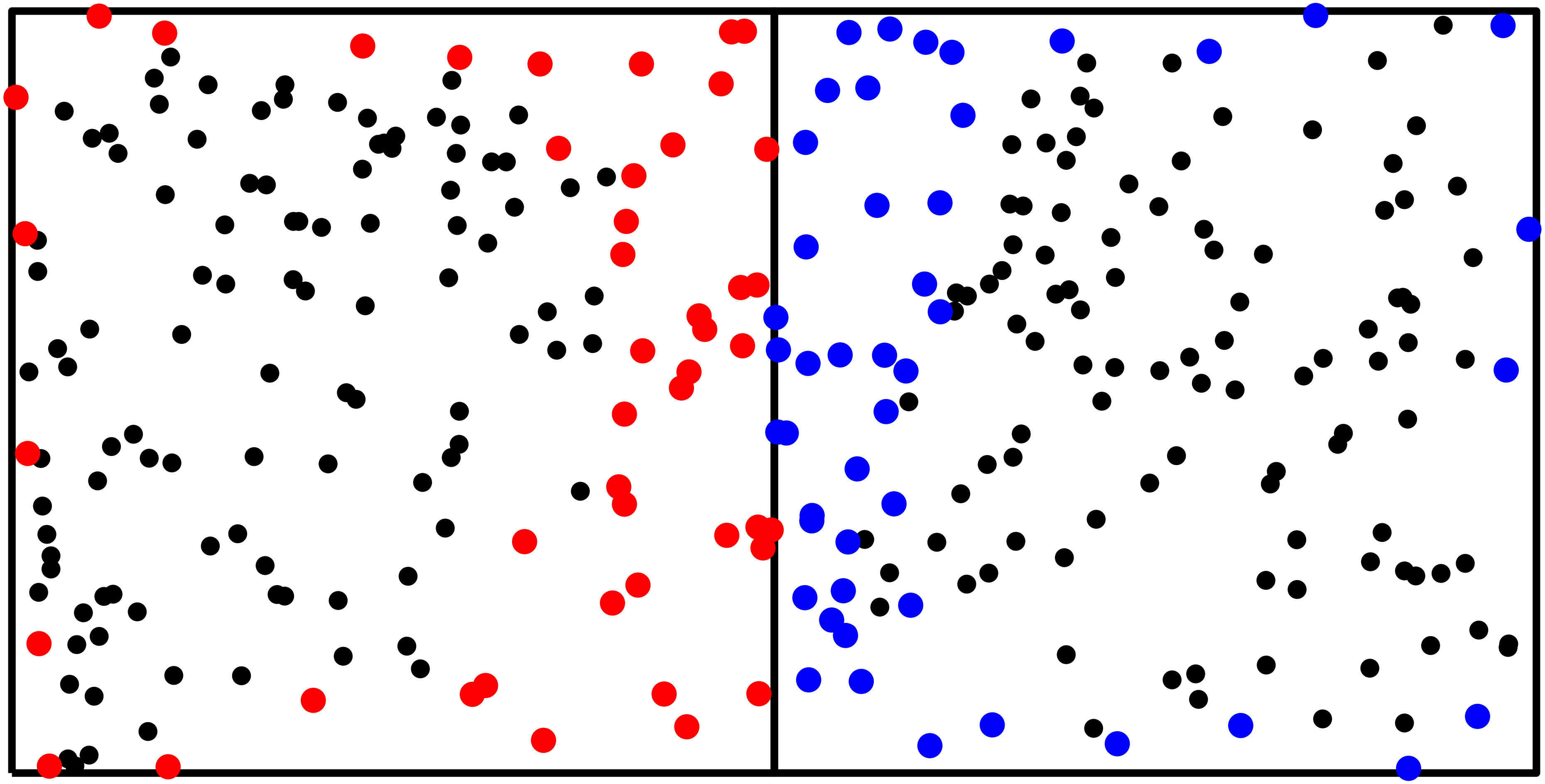


Rank $= 29$ at $\varepsilon = 10^{-10}$.



Rank = 48 at $\varepsilon = 10^{-10}$.

Adjacent boxes can be skeletonized.



Rank $= 46$ at $\varepsilon = 10^{-10}$.

$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[r, "A"] \arrow[d, "U_{\mathrm{s}}^{\mathrm{t}}"'] & \{v_m\}_{m=1}^{M} \\
\{\tilde{q}_{n_j}\}_{j=1}^{k} \arrow[r, "A^{\mathrm{skel}}"'] & \{v_{m_j}\}_{j=1}^{k} \arrow[u, "U_{\mathrm{t}}"']
\end{tikzcd}$$

Benefits:

- The rank is optimal.
- The projection and interpolation are cheap.
  U_{s} and U_{t} contain $k \times k$ identity matrices.
- The projection and interpolation are well-conditioned.
- Finding the k points is inexpensive.
- The matrix $\tilde{A}$ is a submatrix of the original matrix A.
  (We loosely say that "the physics of the problem is preserved".)
- Interaction between **adjacent** boxes can be compressed
  (no buffering is required).

Similar schemes have been proposed by many researchers:

1993 - C.R. Anderson

1995 - C.L. Berman

1996 - E. Michielssen, A. Boag

1999 - J. Makino

2004 - L. Ying, G. Biros, D. Zorin

A mathematical foundation:

1996 - M. Gu, S. Eisenstat

Item 2 (out of 3): Hierarchical computation of an inverse operator

Consider the linear system

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}.$$

We suppose that for $i \neq j$, the blocks A_{ij} allow the factorization

$$\underbrace{A_{ij}}_{n_i \times n_j} = \underbrace{U_i}_{n_i \times k_i} \; \underbrace{\tilde{A}_{ij}}_{k_i \times k_j} \; \underbrace{U_j^{\rm t}}_{k_j \times n_j},$$

where the ranks k_i are significantly smaller than the block sizes n_i.

We then let

$$\underbrace{\tilde{q}_j}_{k_j \times 1} = U_j^{\rm t} \; \underbrace{q_j}_{n_j \times 1},$$

be the variables of the “reduced” system.

Recall:

- $A_{ij} = U_i\,\tilde{A}_{ij}\,U_j^{\rm t}$
- q_j is the variable in the original model — fine scale
- $\tilde{q}_j = U_j^{\rm t}\,q_j$ — coarse scale

The system $\sum_j A_{ij}\,q_j = v_i$ then takes the form

$$\left[\begin{array}{cccc|cccc}
A_{11} & 0 & 0 & 0 & 0 & U_1\tilde{A}_{12} & U_1\tilde{A}_{13} & U_1\tilde{A}_{14} \\
0 & A_{22} & 0 & 0 & U_2\tilde{A}_{21} & 0 & U_2\tilde{A}_{23} & U_2\tilde{A}_{24} \\
0 & 0 & A_{33} & 0 & U_3\tilde{A}_{31} & U_3\tilde{A}_{32} & 0 & U_3\tilde{A}_{34} \\
0 & 0 & 0 & A_{44} & U_4\tilde{A}_{41} & U_4\tilde{A}_{42} & U_4\tilde{A}_{43} & 0 \\
\hline
-U_1^{\rm t} & 0 & 0 & 0 & I & 0 & 0 & 0 \\
0 & -U_2^{\rm t} & 0 & 0 & 0 & I & 0 & 0 \\
0 & 0 & -U_3^{\rm t} & 0 & 0 & 0 & I & 0 \\
0 & 0 & 0 & -U_4^{\rm t} & 0 & 0 & 0 & I
\end{array}\right]
\left[\begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \tilde{q}_1 \\ \tilde{q}_2 \\ \tilde{q}_3 \\ \tilde{q}_4 \end{array}\right]
=
\left[\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right].$$

Now form the Schur complement to eliminate the q_j's.

After eliminating the “fine-scale” variables q_i, we obtain

$$\begin{bmatrix} I & U_1^{\rm t}\tilde{A}_{11}^{-1}U_1\tilde{A}_{12} & U_1^{\rm t}\tilde{A}_{11}^{-1}U_1\tilde{A}_{13} & U_1^{\rm t}\tilde{A}_{11}^{-1}U_1\tilde{A}_{14} \\ U_2^{\rm t}\tilde{A}_{22}^{-1}U_2\tilde{A}_{21} & I & U_2^{\rm t}\tilde{A}_{22}^{-1}U_2\tilde{A}_{23} & U_2^{\rm t}\tilde{A}_{22}^{-1}U_2\tilde{A}_{24} \\ U_3^{\rm t}\tilde{A}_{33}^{-1}U_3\tilde{A}_{31} & U_3^{\rm t}\tilde{A}_{33}^{-1}U_3\tilde{A}_{32} & I & U_3^{\rm t}\tilde{A}_{33}^{-1}U_3\tilde{A}_{34} \\ U_4^{\rm t}\tilde{A}_{44}^{-1}U_4\tilde{A}_{41} & U_4^{\rm t}\tilde{A}_{44}^{-1}U_4\tilde{A}_{42} & U_4^{\rm t}\tilde{A}_{44}^{-1}U_4\tilde{A}_{43} & I \end{bmatrix} \begin{bmatrix} \tilde{q}_1 \\ \tilde{q}_2 \\ \tilde{q}_3 \\ \tilde{q}_4 \end{bmatrix} = \begin{bmatrix} U_1^{\rm t}A_{11}^{-1}\,v_1 \\ U_2^{\rm t}A_{22}^{-1}\,v_2 \\ U_3^{\rm t}A_{33}^{-1}\,v_3 \\ U_4^{\rm t}A_{44}^{-1}\,v_4. \end{bmatrix}.$$

We set

$$\tilde{A}_{ii} = \left(U_i^{\rm t}\,A_{ii}^{-1}\,U_i\right)^{-1},$$

and multiply line i by $\tilde{A}_{ii}$ to obtain the reduced system

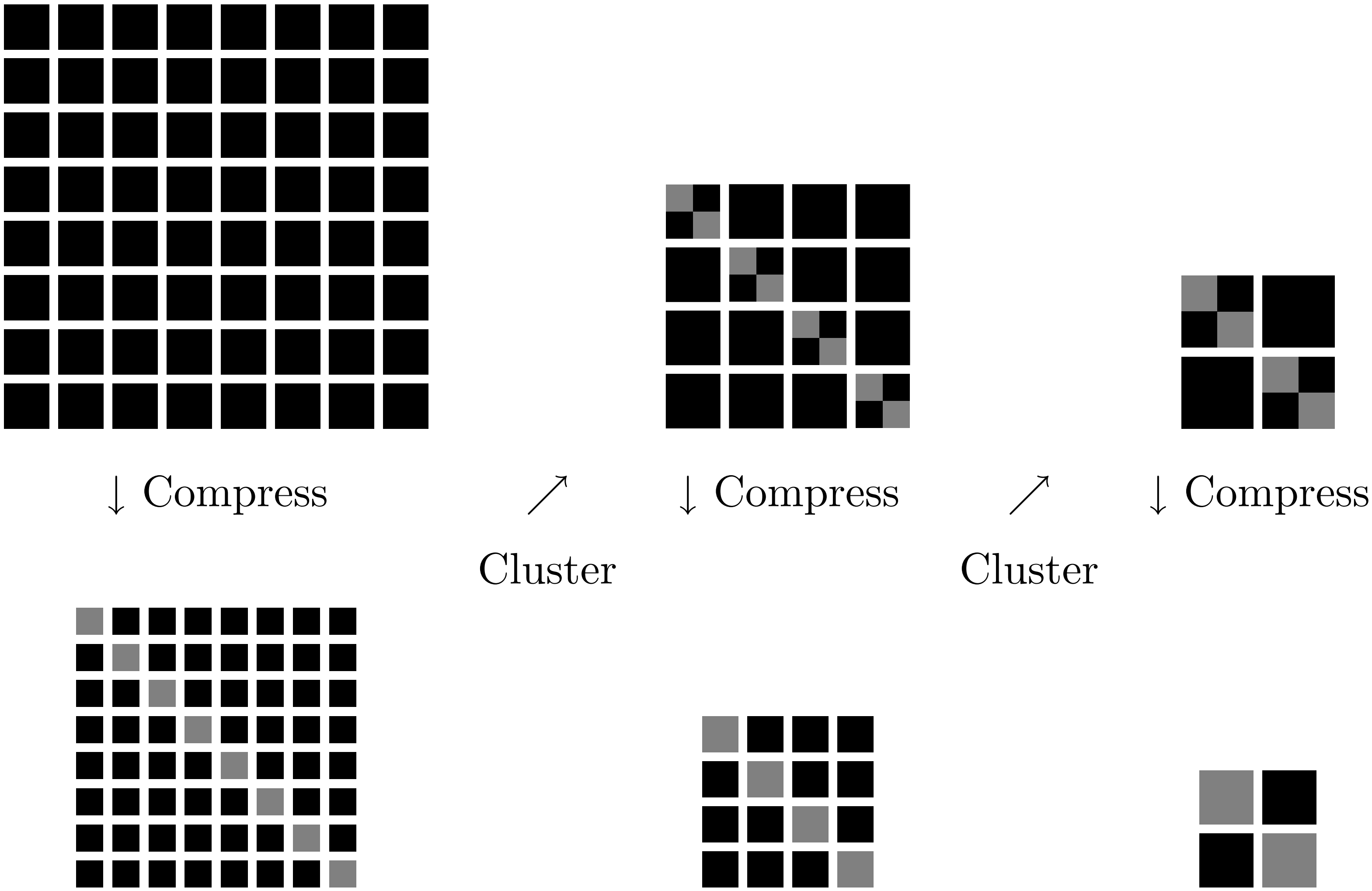
$$\begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} & \tilde{A}_{13} & \tilde{A}_{14} \\ \tilde{A}_{21} & \tilde{A}_{22} & \tilde{A}_{23} & \tilde{A}_{24} \\ \tilde{A}_{31} & \tilde{A}_{32} & \tilde{A}_{33} & \tilde{A}_{34} \\ \tilde{A}_{41} & \tilde{A}_{42} & \tilde{A}_{43} & \tilde{A}_{44} \end{bmatrix} \begin{bmatrix} \tilde{q}_1 \\ \tilde{q}_2 \\ \tilde{q}_3 \\ \tilde{q}_4 \end{bmatrix} = \begin{bmatrix} \tilde{v}_1 \\ \tilde{v}_2 \\ \tilde{v}_3 \\ \tilde{v}_4 \end{bmatrix}.$$

where

$$\tilde{v}_i = \tilde{A}_{ii}\,U_i^{\rm t}\,A_{ii}^{-1}\,v_i.$$

(This derivation was pointed out by Leslie Greengard.)

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



↓ Compress ↗ Cluster ↓ Compress ↗ Cluster ↓ Compress

Recall that the one-level compression scheme requires the construction of approximate factorization of the off-diagonal blocks:

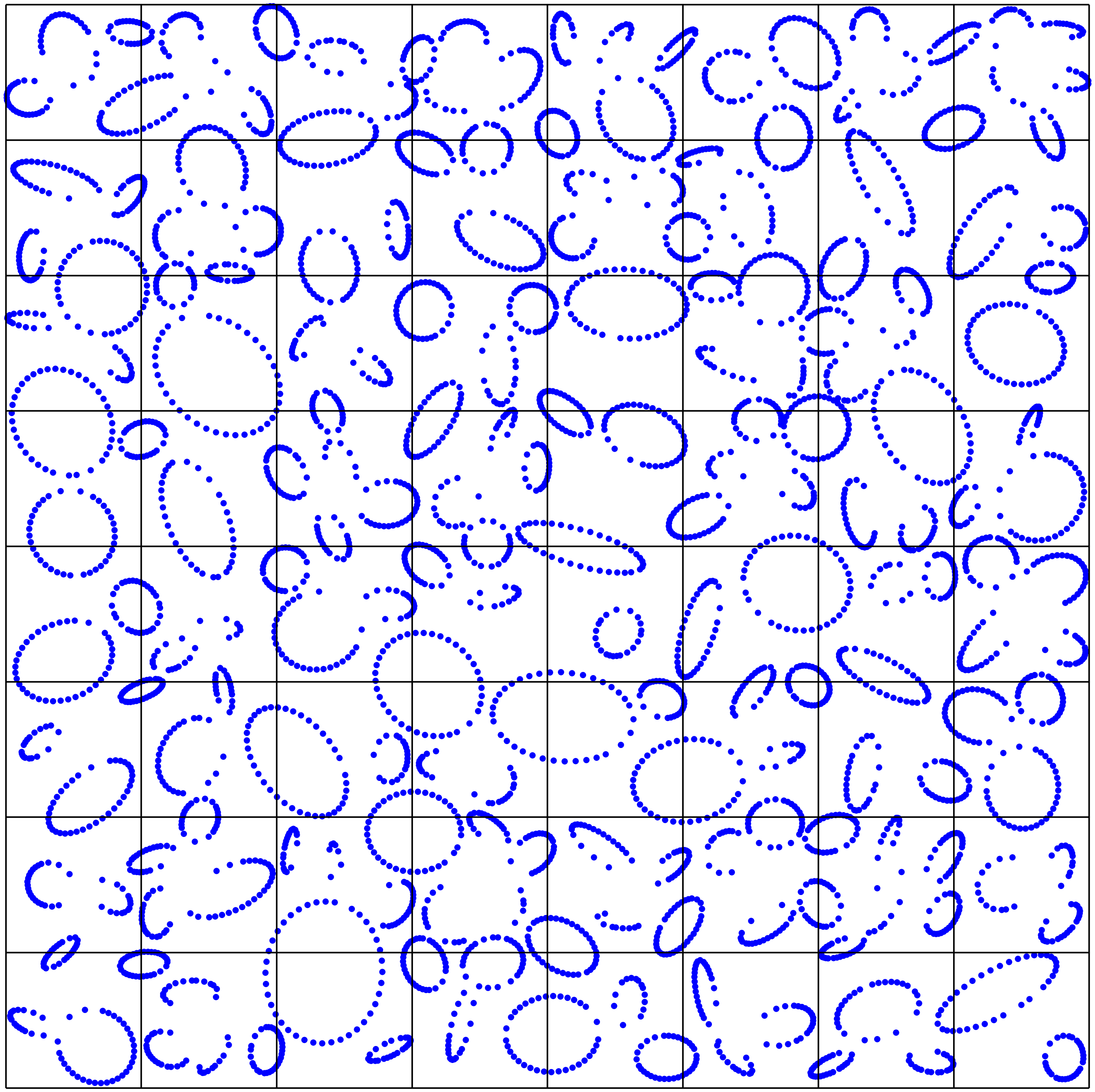
$$A_{ij} = U_i\,\tilde{A}_{ij}\,U_j^{\mathrm{t}},$$

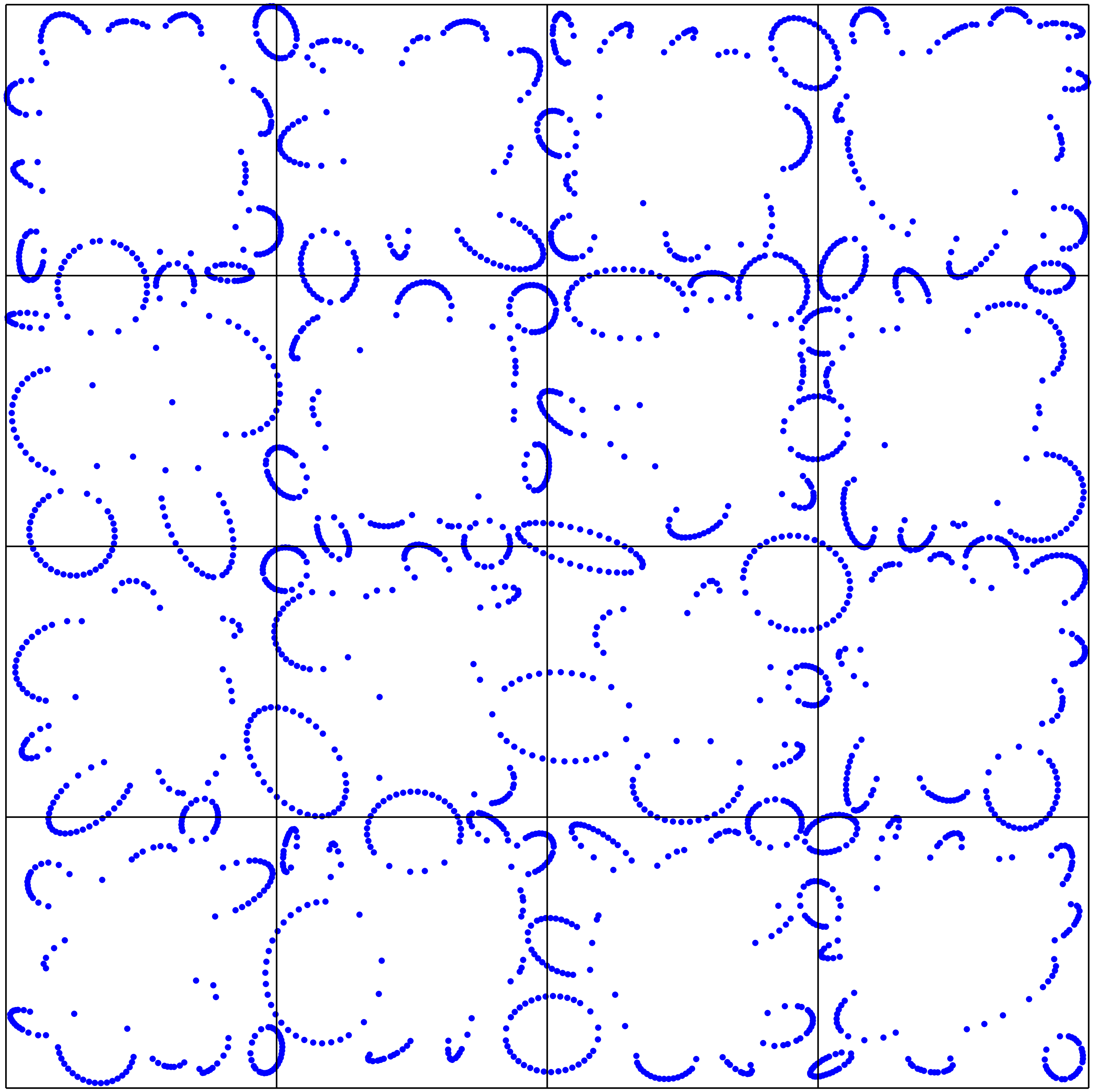
When the interpolative decompositions are used, the matrices $\tilde{A}_{ij}$ will be submatrices of the original matrices A_{ij}.

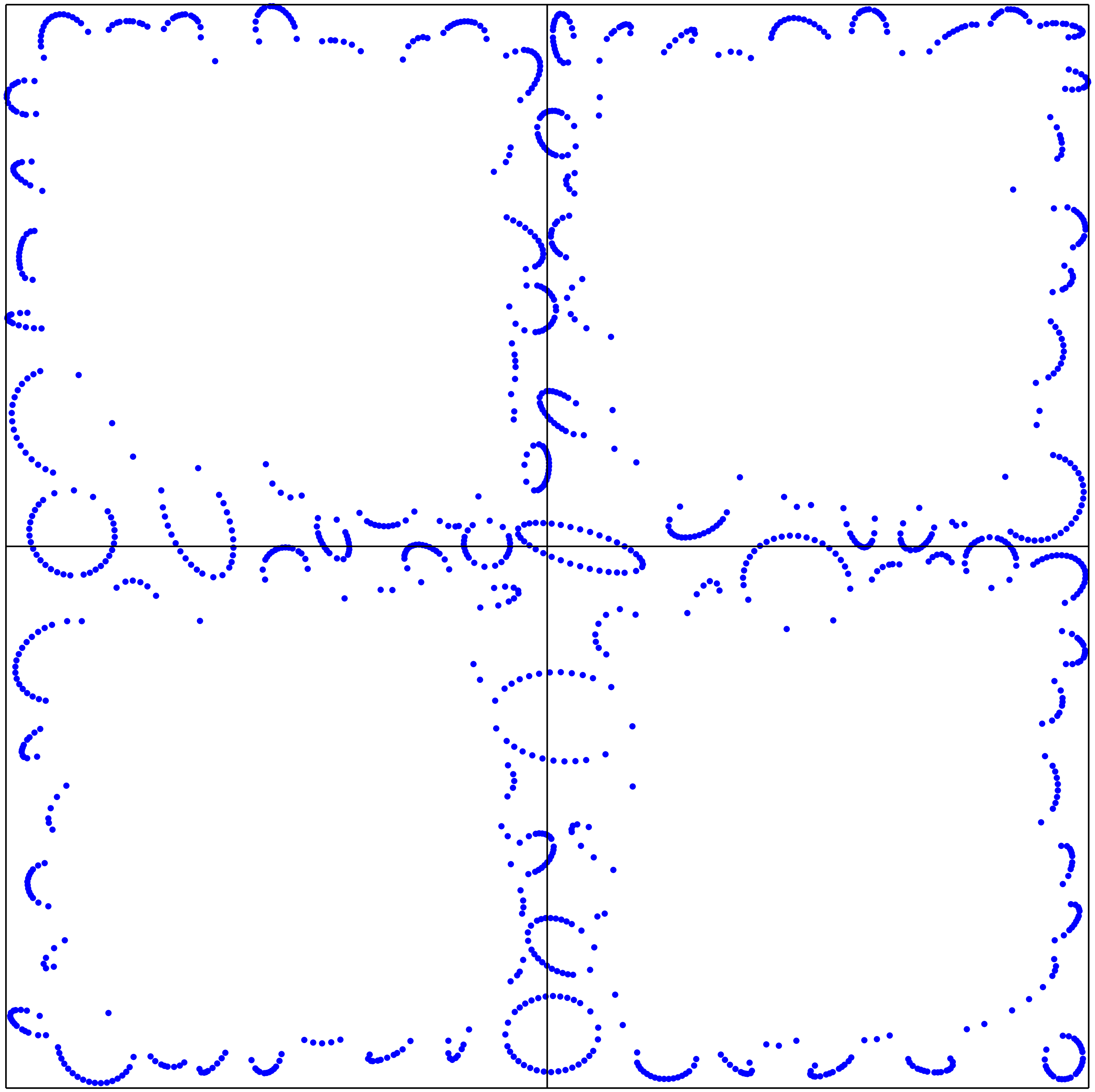
This leads to dramatic cost savings since the $\tilde{A}_{ij}$'s need never be explicitly computed.

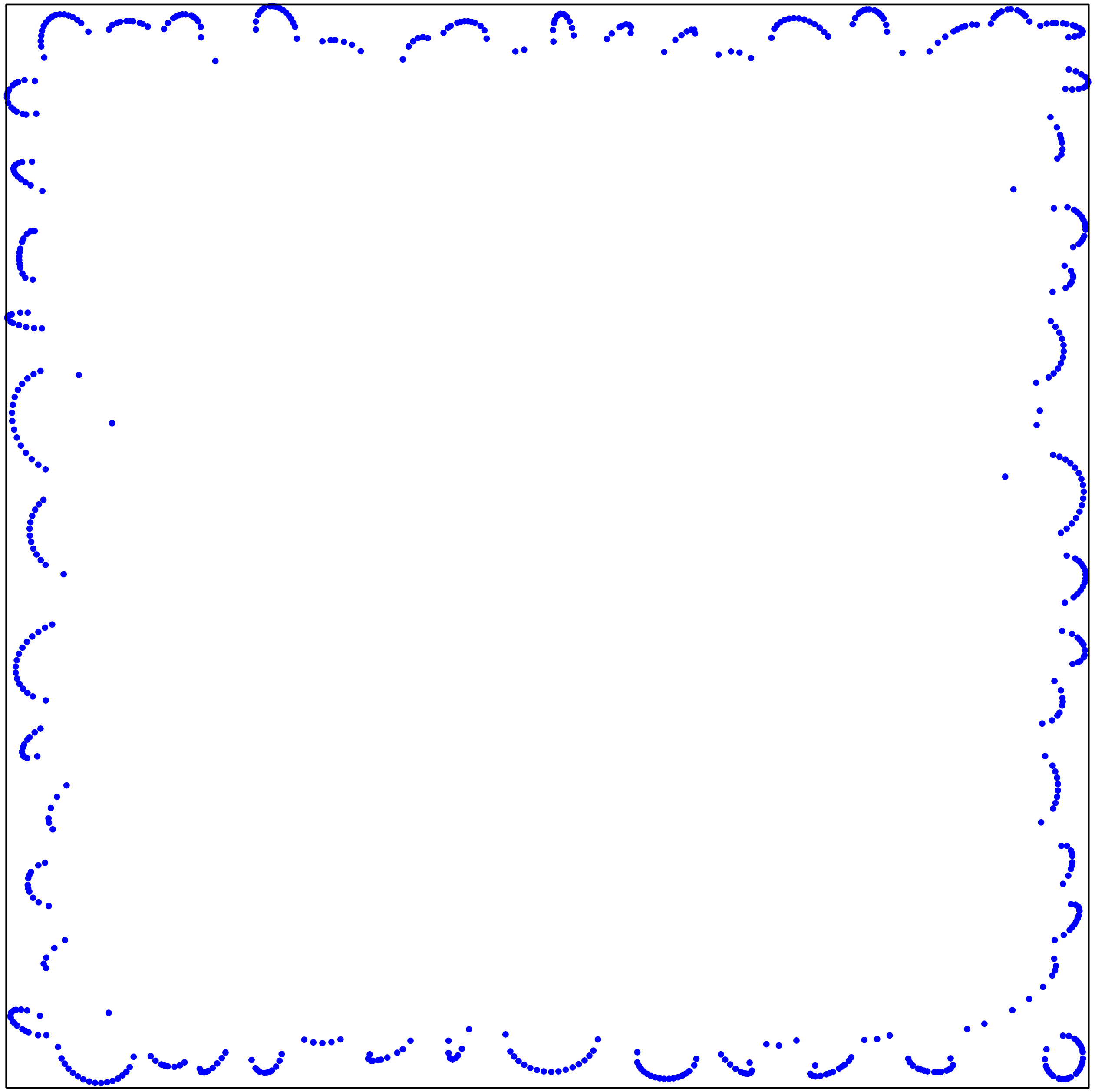
Another advantage of using interpolative decompositions is that each degree of freedom at any level is associated with a distinct grid point.

In other words, in going from one level to the next coarser one, we in effect throw out a number of grid-points.









Recall — again — that the one-level compression scheme requires the construction of approximate factorizations of the off-diagonal blocks:

$$A_{ij} = U_i\,\tilde{A}_{ij}\,U_j^{\rm t},$$

Note that each matrix of basis vectors U_i needs to be a basis for *every off-diagonal block in i'th row of blocks* (and the i'th column of blocks as well).

There are different approaches to solving this seemingly very expensive task:

- Use analytic expansions of the kernel [original FMM].
  - Impossible when inverting operators. (The kernel is à priori not known!)
- Interpolation of the kernel function [Hackbusch, BCR, etc].
  - Requires estimates of smoothness of the kernel away from the diagonal.
  - Inefficient, does not work for all geometries.
- Green's identities satisfied by the kernel [M./Rokhlin, Biros/Ying/Zorin, *etc*].
  - Very robust.
  - Leads to representations that are very close to optimal.
- Randomized sampling. New!

Recall: We convert the system

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

Fine resolution.
Large blocks.

to the reduced system

$$\begin{bmatrix} \tilde{A}_{11} & A_{12}^{\mathrm{skel}} & A_{13}^{\mathrm{skel}} & A_{14}^{\mathrm{skel}} \\ A_{21}^{\mathrm{skel}} & \tilde{A}_{22} & A_{23}^{\mathrm{skel}} & A_{24}^{\mathrm{skel}} \\ A_{31}^{\mathrm{skel}} & A_{32}^{\mathrm{skel}} & \tilde{A}_{33} & A_{34}^{\mathrm{skel}} \\ A_{41}^{\mathrm{skel}} & A_{42}^{\mathrm{skel}} & A_{43}^{\mathrm{skel}} & \tilde{A}_{44} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \end{bmatrix} = \begin{bmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \tilde{f}_3 \\ \tilde{f}_4 \end{bmatrix}$$

Coarse resolution.
Small blocks.

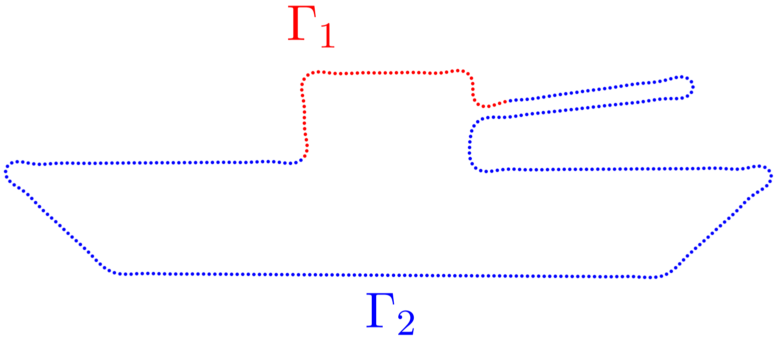
where A_{ij}^{skel} is a submatrix of A_{ij} when $i \neq j$.

Note that in the one-level compression, the only objects actually computed are the index vectors that identify the sub-matrices, and the new diagonal blocks $\tilde{A}_{ii}$.

What are the blocks $\tilde{A}_{ii}$?

We recall that the new diagonal blocks are defined by

$$\underbrace{\tilde{A}_{ii}}_{k\times k} = \big(\underbrace{U_i^{\mathrm{t}}}_{k\times n}\,\underbrace{A_{ii}^{-1}}_{n\times n}\,\underbrace{U_i}_{n\times k}\big)^{-1}.$$



We call these blocks "proxy matrices".

What are they?

Let Γ_1 denote the block marked in red.

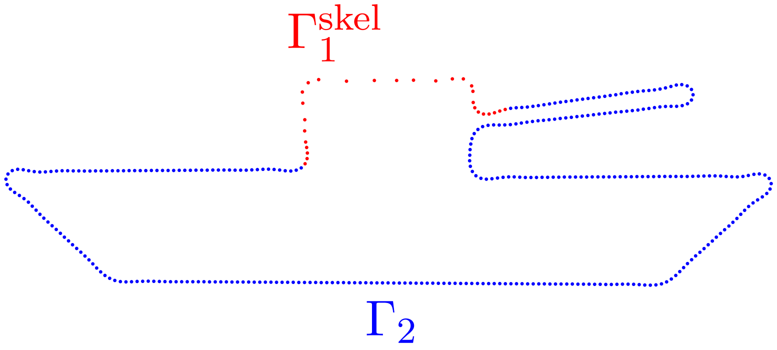
Let Γ_2 denote the rest of the domain.

$$\begin{tikzcd}
\text{Charges on } \Gamma_2 \arrow[r, "A_{12}"] \arrow[dr, "A_{12}^{\mathrm{skel}}"'] & \text{Pot. on } \Gamma_1 \arrow[r, "A_{11}^{-1}"] & \text{Charges on } \Gamma_1 \arrow[r, "A_{21}"] \arrow[d, "U_1^{\mathrm{t}}"'] & \text{Pot. on } \Gamma_2 \\
& \text{Pot. on } \Gamma_1^{\mathrm{skel}} \arrow[u, "U_1"'] \arrow[r, "\tilde{A}_{11}^{-1}"] & \text{Charges on } \Gamma_1^{\mathrm{skel}} \arrow[ur, "A_{21}^{\mathrm{skel}}"'] &
\end{tikzcd}$$

$\tilde{A}_{11}$ contains *all the information the outside world needs to know about* Γ_1.

We recall that the new diagonal blocks are defined by

$$\underbrace{\tilde{A}_{ii}}_{k\times k} = \big(\underbrace{U_i^{\rm t}}_{k\times n}\,\underbrace{A_{ii}^{-1}}_{n\times n}\,\underbrace{U_i}_{n\times k}\big)^{-1}.$$



We call these blocks "proxy matrices".

What are they?

Let Γ_1 denote the block marked in red.

Let Γ_2 denote the rest of the domain.

$$\begin{tikzcd}
\text{Charges on } \Gamma_2 \arrow[r, "A_{12}"] \arrow[dr, "A_{12}^{\rm skel}"'] & \text{Pot. on } \Gamma_1 \arrow[r, "A_{11}^{-1}"] & \text{Charges on } \Gamma_1 \arrow[r, "A_{21}"] \arrow[d, "U_1^{\rm t}"'] & \text{Pot. on } \Gamma_2 \\
 & \text{Pot. on } \Gamma_1^{\rm skel} \arrow[u, "U_1"'] \arrow[r, "\tilde{A}_{11}^{-1}"] & \text{Charges on } \Gamma_1^{\rm skel} \arrow[ur, "A_{21}^{\rm skel}"'] &
\end{tikzcd}$$

$\tilde{A}_{11}$ contains *all the information the outside world needs to know about* Γ_1.

We recall that the new diagonal blocks are defined by

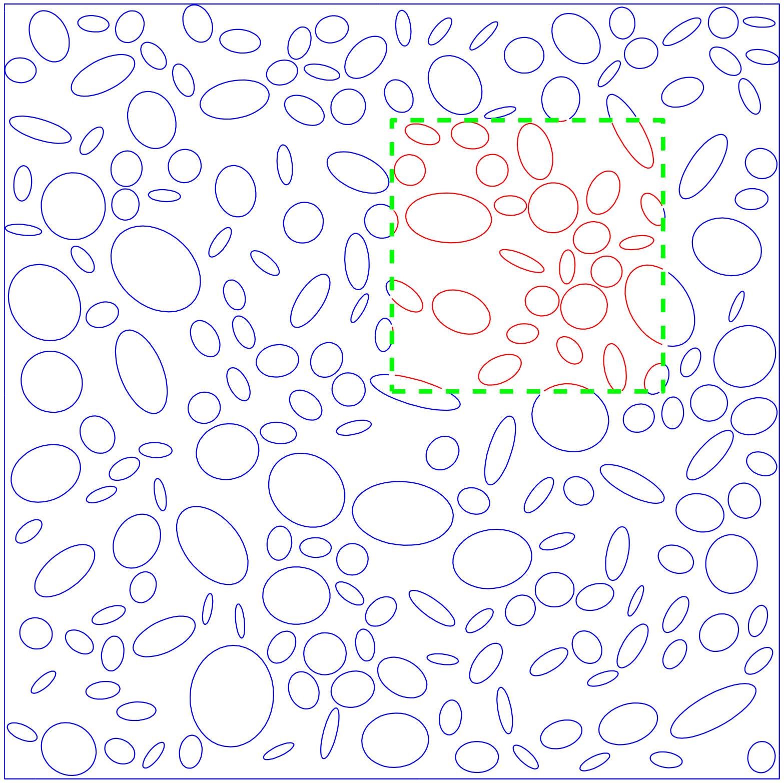
$$\underbrace{\tilde{A}_{ii}}_{k\times k} = \big(\underbrace{U_i^{\mathrm{t}}}_{k\times n}\,\underbrace{A_{ii}^{-1}}_{n\times n}\,\underbrace{U_i}_{n\times k}\big)^{-1}.$$

We call these blocks "proxy matrices".

What are they?

Let Ω_1 denote the block marked in red.

Let Ω_2 denote the rest of the domain.



$$\begin{tikzcd}
\text{Charges on } \Omega_2 \arrow[r, "A_{12}"] \arrow[dr, "A_{12}^{\mathrm{skel}}"'] & \text{Pot. on } \Omega_1 \arrow[r, "A_{11}^{-1}"] & \text{Charges on } \Omega_1 \arrow[r, "A_{21}"] \arrow[d, "U_1^{\mathrm{t}}"'] & \text{Pot. on } \Omega_2 \\
& \text{Pot. on } \Omega_1^{\mathrm{skel}} \arrow[u, "U_1"'] \arrow[r, "\tilde{A}_{11}^{-1}"] & \text{Charges on } \Omega_1^{\mathrm{skel}} \arrow[ur, "A_{21}^{\mathrm{skel}}"'] &
\end{tikzcd}$$

$\tilde{A}_{11}$ contains *all the information the outside world needs to know about* Ω_1.

We recall that the new diagonal blocks are defined by

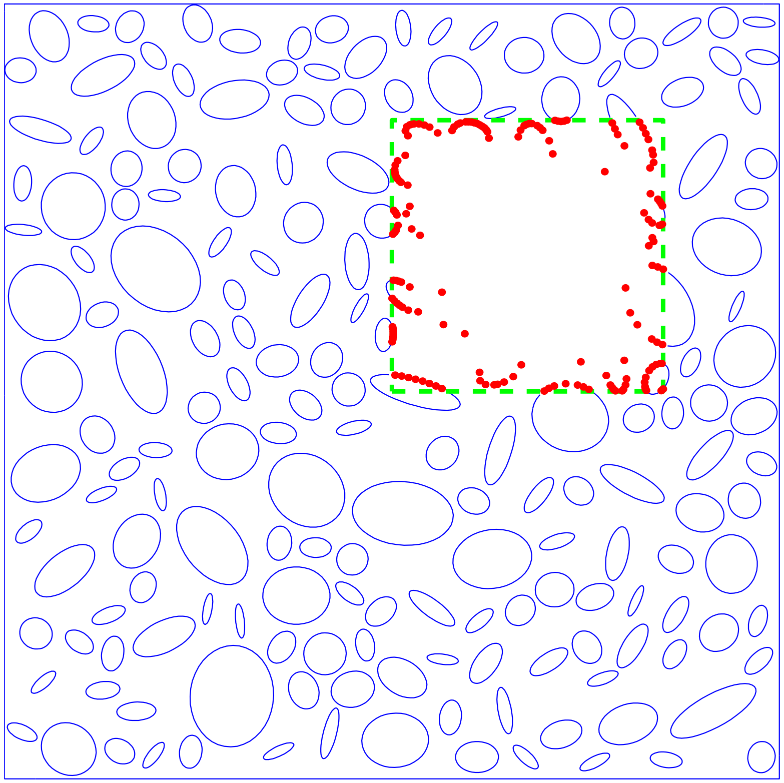
$$\underbrace{\tilde{A}_{ii}}_{k\times k} = \big(\underbrace{U_i^{\rm t}}_{k\times n}\,\underbrace{A_{ii}^{-1}}_{n\times n}\,\underbrace{U_i}_{n\times k}\big)^{-1}.$$

We call these blocks “proxy matrices”.

What are they?

Let Ω_1 denote the block marked in red.

Let Ω_2 denote the rest of the domain.



$$\begin{tikzcd}
\text{Charges on } \Omega_2 \arrow[r, "A_{12}"] \arrow[dr, "A_{12}^{\rm skel}"'] & \text{Pot. on } \Omega_1 \arrow[r, "A_{11}^{-1}"] & \text{Charges on } \Omega_1 \arrow[r, "A_{21}"] \arrow[d, "U_1^{\rm t}"'] & \text{Pot. on } \Omega_2 \\
 & \text{Pot. on } \Omega_1^{\rm skel} \arrow[u, "U_1"'] \arrow[r, "\tilde{A}_{11}^{-1}"] & \text{Charges on } \Omega_1^{\rm skel} \arrow[ur, "A_{21}^{\rm skel}"'] &
\end{tikzcd}$$

$\tilde{A}_{11}$ contains *all the information the outside world needs to know about Ω_1*.

Item 3 (out of 3):
Approximation of matrices using randomized sampling

As we have seen, a core component of these fast schemes concerns the approximation of matrices by low-rank approximations.

This computational task can be characterized as follows:

- We are given a (potentially large) matrix A that we know can be approximated well by a low-rank matrix.

- We have at our disposal fast methods for evaluating the map $x \mapsto A\,x$.

- We seek to rapidly construct an approximate basis for the column space of A. In other words, we seek to construct a matrix $V = [v_1,\, v_2,\, \dots,\, v_k]$ with orthonormal columns such that

  $$||A - V\,V^{\rm t}\,A|| \leq \varepsilon,$$

  where ε is a given computational accuracy.

Algorithm:

Rapid computation of a low-rank appoximation.

- Let ε denote the computational accuracy desired.
- Let A be an $m \times n$ matrix of ε-rank k.
- We seek a rank-k approximation of A.
- We can perform matrix-vector multiplies fast.

Let $\omega_1, \omega_2, \dots$ be a sequence of vectors in $\mathbb{R}^n$ whose entries are i.i.d. random variables drawn from a standardized Gaussian distribution.

Form the length-m vectors

$$y_1 = A\,\omega_1, \qquad y_2 = A\,\omega_2, \qquad y_3 = A\,\omega_3, \qquad \dots$$

Each y_j is a “random linear combination” of columns of A.

If l is an integer such that $l \geq k$, then there is a chance that the vectors

$$\{y_1, y_2, \dots, y_l\}$$

span the column space of A “to within precision ε”. Clearly, the probability that this happens gets larger, the larger the gap between l and k.

Algorithm:

Rapid computation of a low-rank appoximation.

- Let ε denote the computational accuracy desired.
- Let A be an $m \times n$ matrix of ε-rank k.
- We seek a rank-k approximation of A.
- We can perform matrix-vector multiplies fast.

Let $\omega_1, \omega_2, \dots$ be a sequence of vectors in $\mathbb{R}^n$ whose entries are i.i.d. random variables drawn from a standardized Gaussian distribution.

Form the length-m vectors

$$y_1 = A\,\omega_1, \qquad y_2 = A\,\omega_2, \qquad y_3 = A\,\omega_3, \qquad \dots$$

Each y_j is a “random linear combination” of columns of A.

If l is an integer such that $l \geq k$, then there is a chance that the vectors

$$\{y_1,\, y_2,\, \dots\,, y_l\}$$

span the column space of A “to within precision ε”. Clearly, the probability that this happens gets larger, the larger the gap between l and k.

What is remarkable is how fast this probability approaches one.

How to measure "how well we are doing":

Let $\omega_1,\,\omega_2,\,\dots,\,\omega_l$ be the sequence of Gaussian random vectors in $\mathbb{R}^n$.

Set $Y_l = [y_1,\,y_2,\,\dots\,,\,y_l] = [A\,\omega_1,\,A\,\omega_2,\,\dots\,,\,A\,\omega_l]$.

Orthogonalize the vectors $[y_1,\,y_2,\,\dots\,,\,y_l]$ and collect the result in the matrix V_l.

The "error" after l steps is then

$$e_l = ||(I - V_l\,V_l^{\rm t})\,A||.$$

In reality, computing e_l is not affordable. Instead, we compute something like

$$f_l = \max_{1\leq j\leq 10} \big|\big|\big(I - V_l\,V_l^{\rm t}\big)\,y_{l+j}\big|\big|.$$

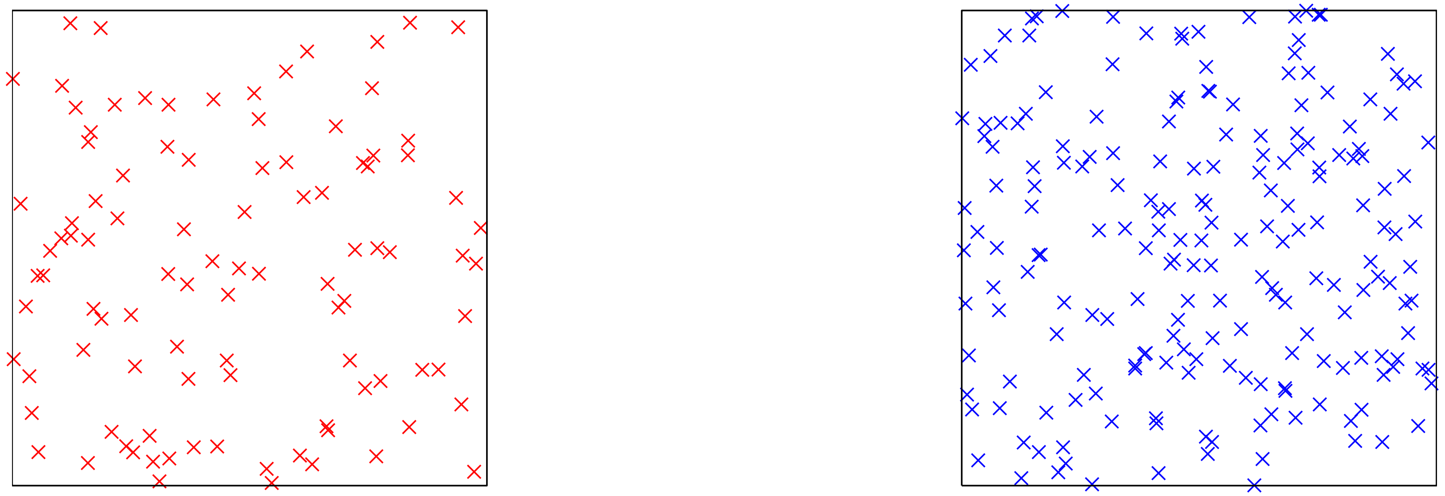
The computation stops when we come to an l such that $f_l < \varepsilon/\sqrt{m}$.

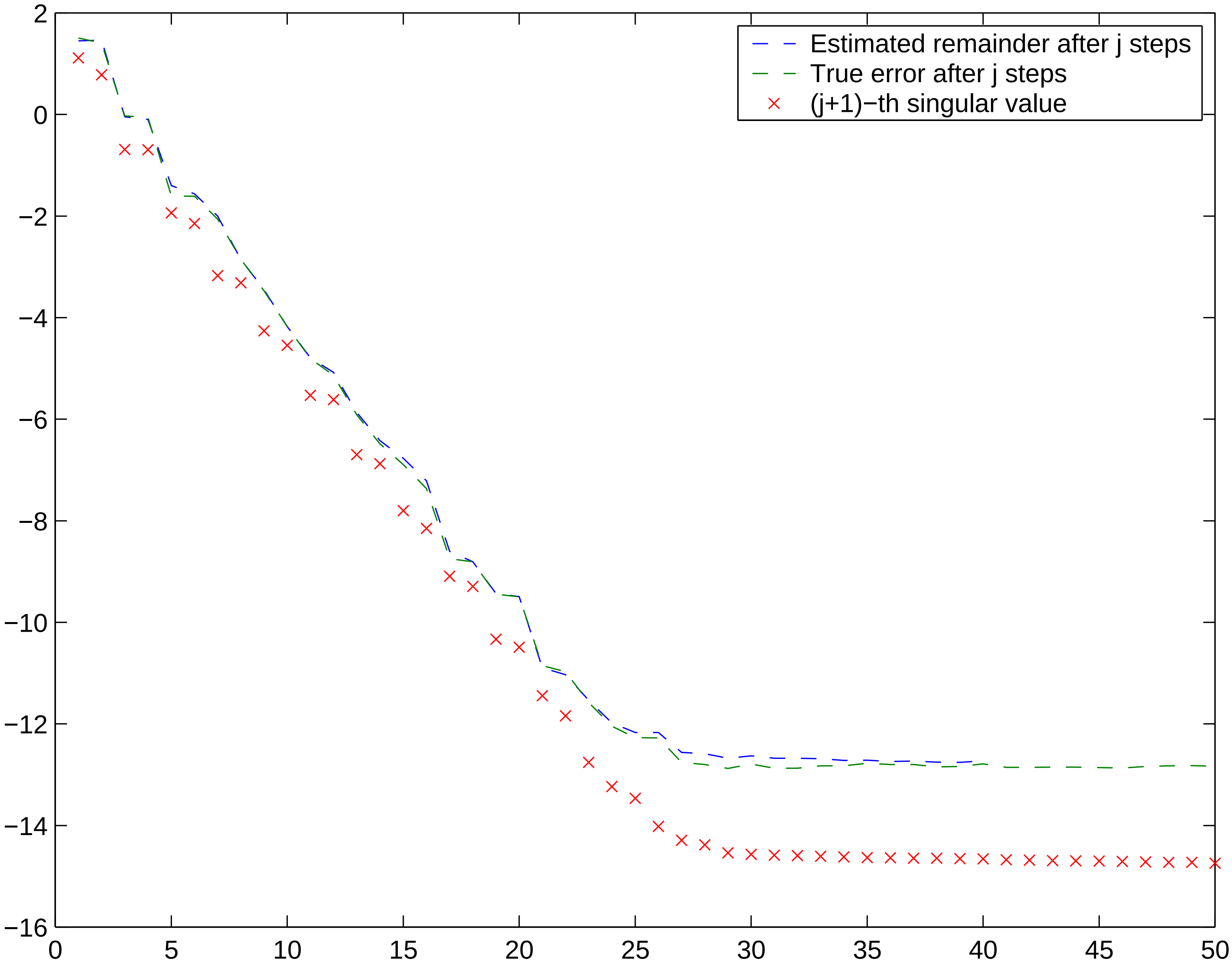
The quantity e_l (estimated by f_l) should be compared to the minimal error

$$\sigma_{l+1} = \min_{\mathrm{rank}(B)=l} ||A - B||.$$

To illustrate the performance, we use the same potential evaluation map A that we saw earlier:

Let A be an $M \times N$ matrix with entries $A_{mn} = \log|x_m - y_n|$ where x_m and y_n are points in two separated clusters in $\mathbb{R}^2$.





$\varepsilon = 10^{-10}$, exact ε-rank $= 34$, nr. of matrix-vector multiplies required $= 36$.

Was this just a lucky realization?

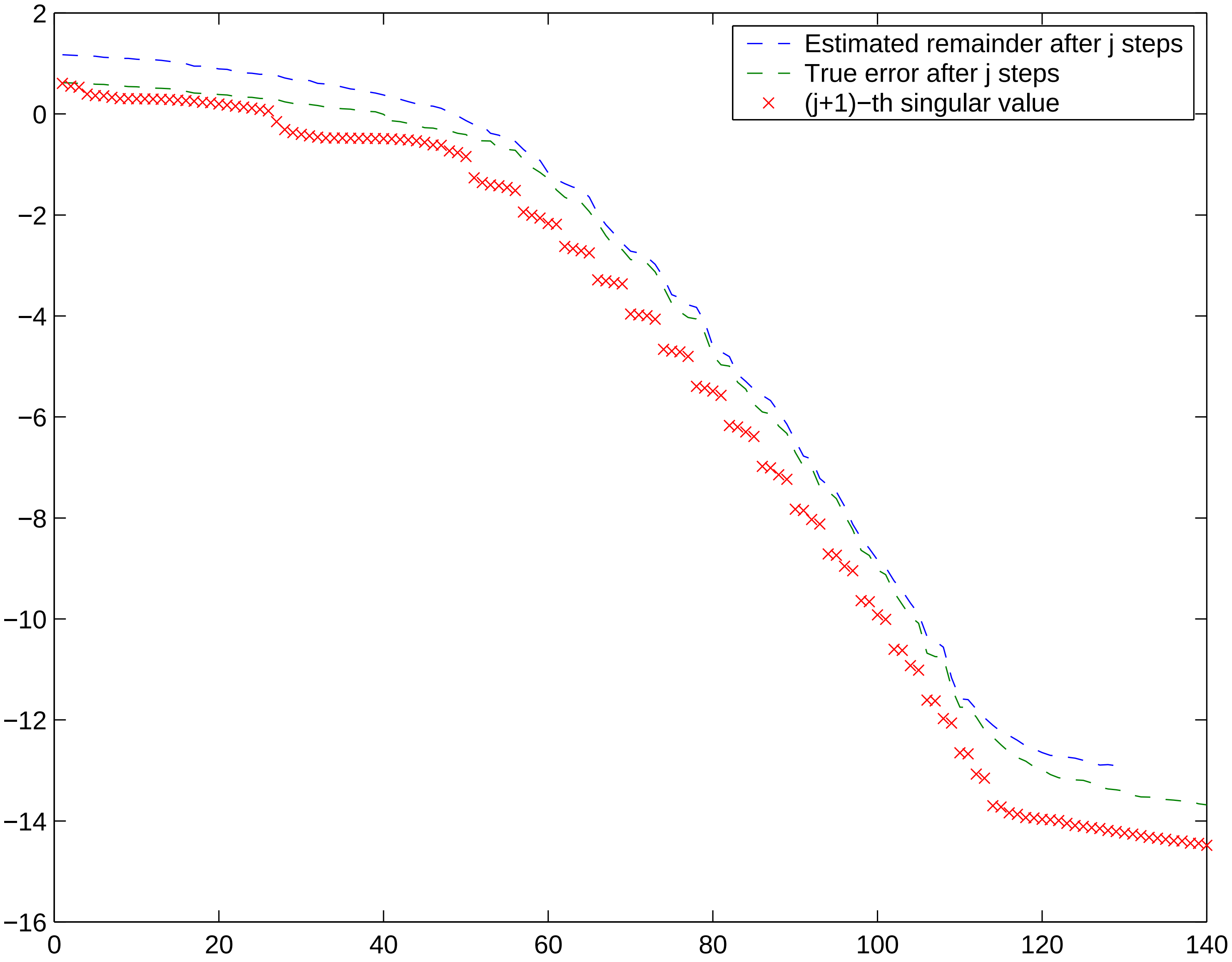
We collected statistics from 1 000 000 realizations:

(Recall that the exact ε-rank is 34.)

Number of matrix-vector multiplies required:	Frequency:
34 (+10)	15063
35 (+10)	376163
36 (+10)	485124
37 (+10)	113928
38 (+10)	9420
39 (+10)	299
40 (+10)	3

Note: The post-processing correctly determined the rank to be 34 *every time*, and the error in the factorization was *always* less than 10^{-10}.

Results from a high-frequency Helmholtz problem (complex arithmetic):



$\varepsilon = 10^{-10}$, exact ε-rank $= 101$, nr. of matrix-vector multiplies required $= 106$.

Theorem: *Let A be an $m \times n$ matrix and let k be an integer.*

Let l be an integer such that $l \geq k$.

Let Ω be an $n \times l$ matrix with i.i.d. Gaussian elements.

Let V be an $m \times l$ matrix whose columns form an ON-basis for the columns of $A\,\Omega$.

Set $\quad \sigma_{k+1} = \min_{\text{rank}(B)=k} ||A - B||.$

Then

$$||A - V\,V^{\text{t}}\,A||_2 \leq 10\,\sqrt{l\,m}\;\sigma_{k+1},$$

with probability at least

$$1 - \varphi(l-k),$$

where φ is a decreasing function satisfying

$$\varphi(8) < 10^{-5}$$

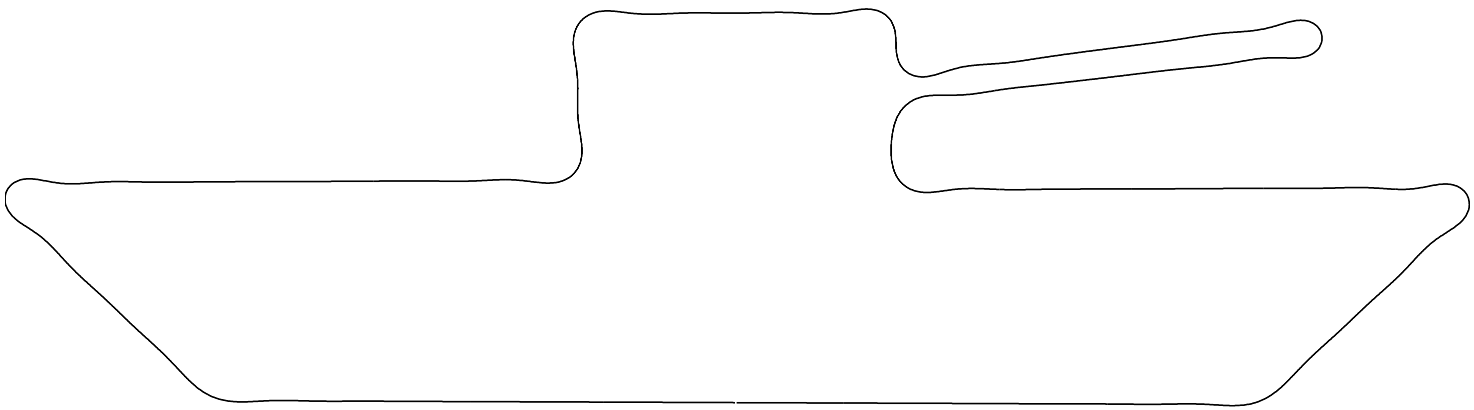
$$\varphi(20) < 10^{-17}.$$

Numerical examples

In developing direct solvers, the “proof is in the pudding” — recall that from a theoretical point of view, the problem is already solved (by Hackbusch and others).

All computations were performed on standard laptops and desktop computers in the 2.0GHz - 2.8Ghz speed range, and with 512Mb of RAM.

An exterior Helmholtz Dirichlet problem



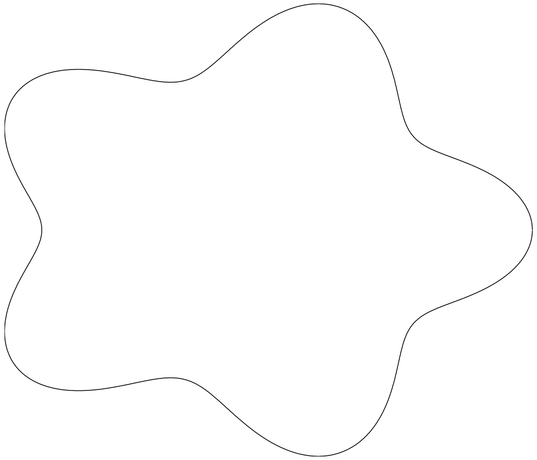
A smooth contour. Its length is roughly 15 and its horizontal width is 2.

k	N_{start}	N_{final}	t_{tot}	t_{solve}	E_{res}	E_{pot}	σ_{min}	M
21	800	435	1.5e+01	3.3e-02	9.7e-08	7.1e-07	6.5e-01	12758
40	1600	550	3.0e+01	6.7e-02	6.2e-08	4.0e-08	8.0e-01	25372
79	3200	683	5.3e+01	1.2e-01	5.3e-08	3.8e-08	3.4e-01	44993
158	6400	870	9.2e+01	2.0e-01	3.9e-08	2.9e-08	3.4e-01	81679
316	12800	1179	1.8e+02	3.9e-01	2.3e-08	2.0e-08	3.4e-01	160493
632	25600	1753	4.3e+02	8.0e-01	1.7e-08	1.4e-08	3.3e-01	350984

Computational results for an exterior Helmholtz Dirichlet problem discretized with 10th order accurate quadrature. The Helmholtz parameter was chosen to keep the number of discretization points per wavelength constant at roughly 45 points per wavelength (resulting in a quadrature error about 10^{-12}).

Eventually . . . the complexity is $O(n + k^3)$.

Example 2 - An interior Helmholtz Dirichlet problem

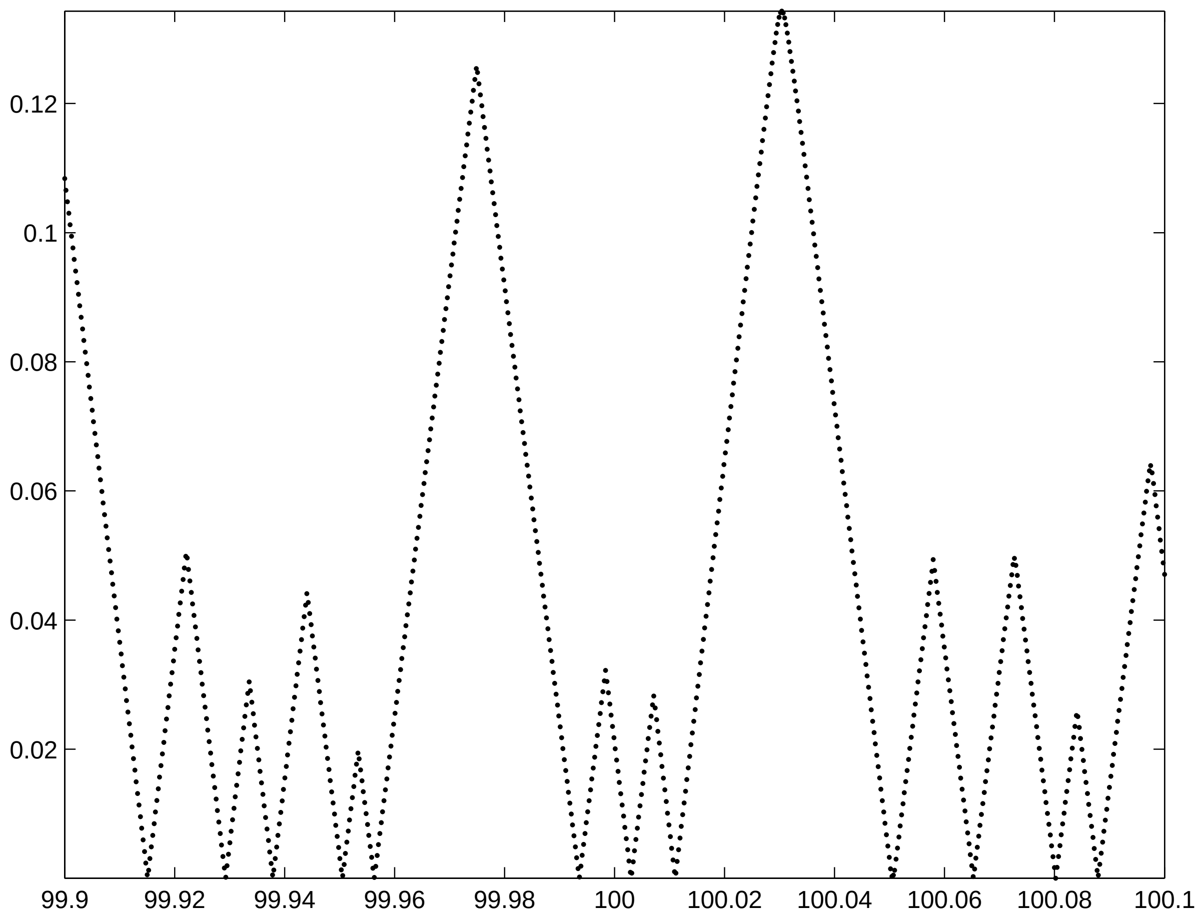


The diameter of the contour is about 2.5. An interior Helmholtz problem with Dirichlet boundary data was solved using $N = 6\,400$ discretization points, with a prescribed accuracy of 10^{-10}.

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.

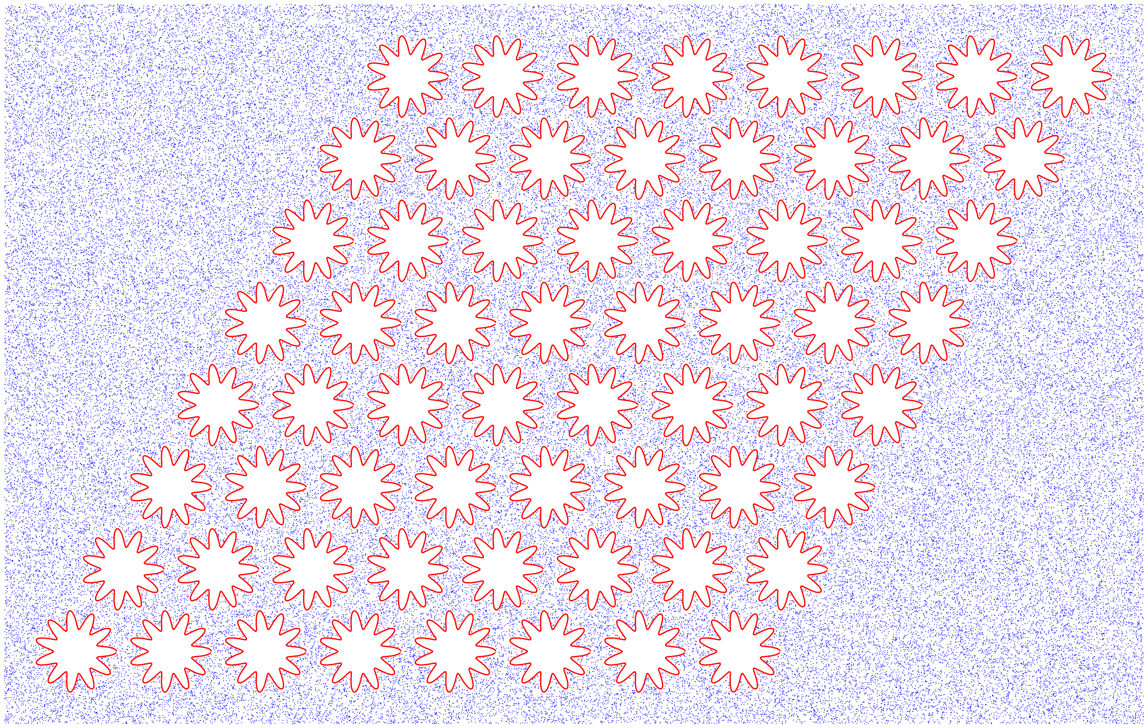
Error in the inverse: 10^{-5}.



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.

Example 3:

An electrostatics problem in a dielectrically heterogeneous medium

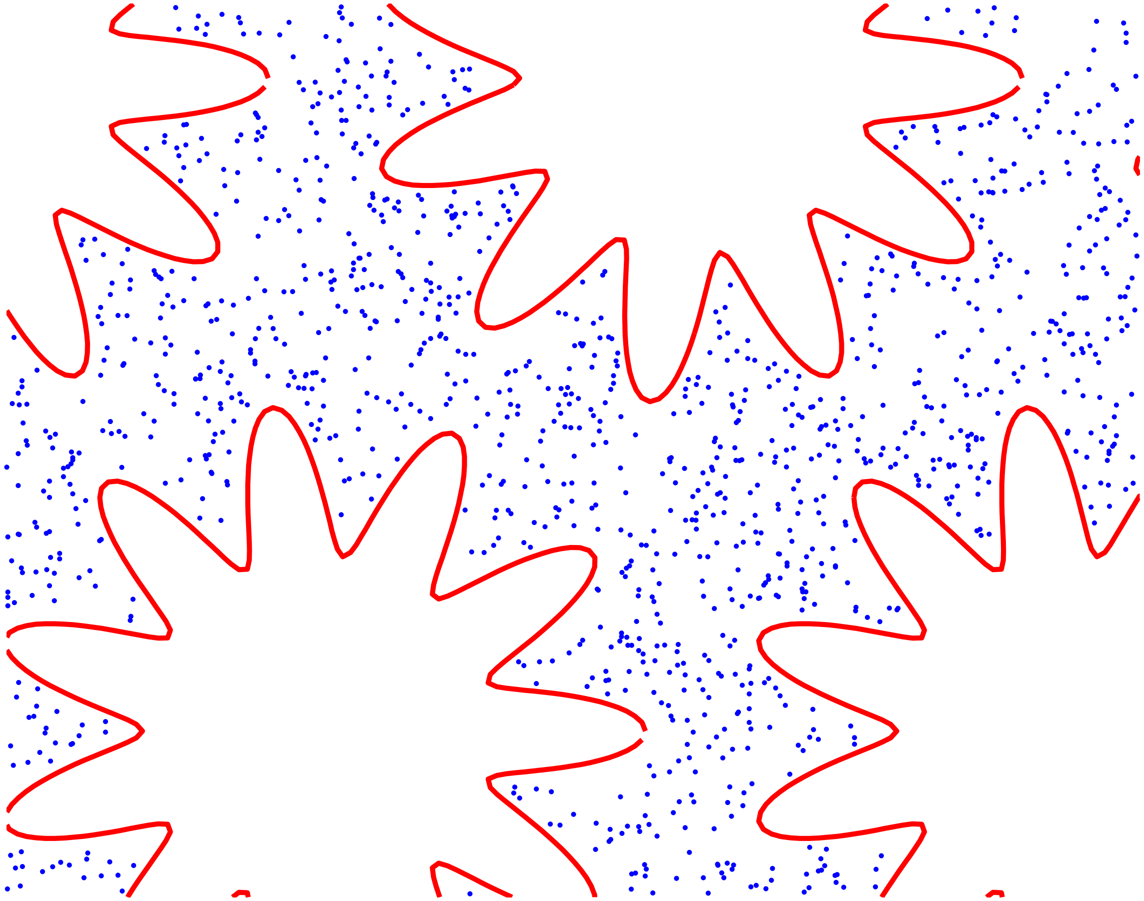


$\varepsilon = 10^{-5}$ $N_{\text{contour}} = 25\,600$ $N_{\text{particles}} = 100\,000$

Time to invert the boundary integral equation = 46sec.

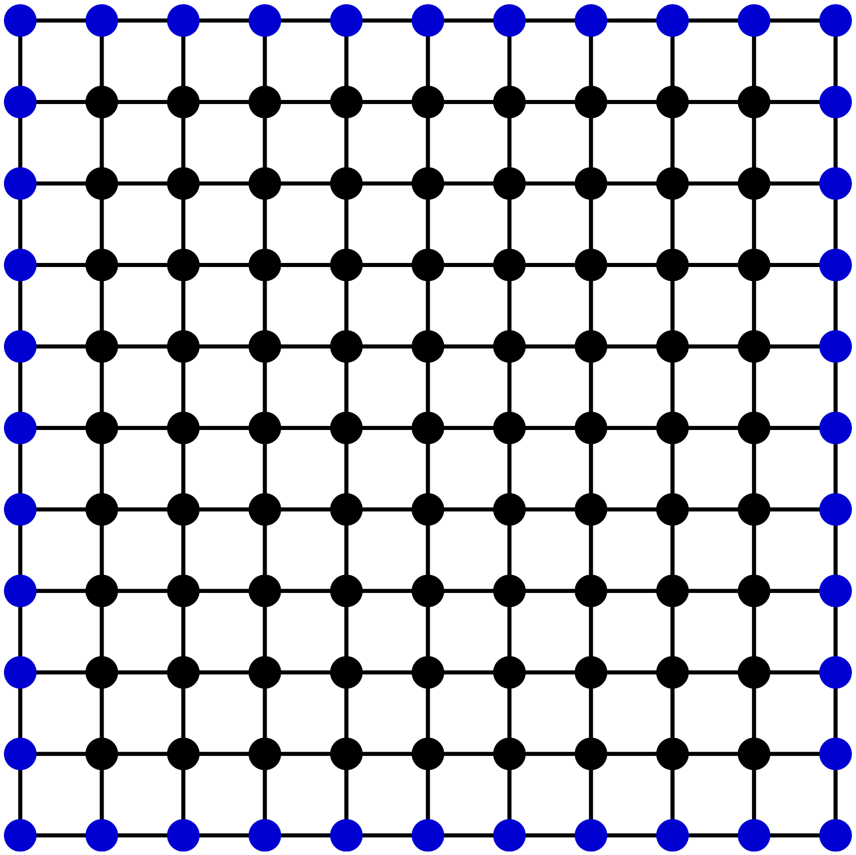
Time to compute the induced charges = 0.42sec.(2.5sec for the FMM)

Total time for the electro-statics problem = 3.8sec.



A close-up of the particle distribution.

Example 4: Inversion of a “Finite Element Matrix”



A grid conduction problem (the “five-point stencil”).

The conductivity of each bar is a random number drawn from a uniform distribution on [1, 2].

If all conductivities were one, then we would get the standard five-point stencil:

$$A = \begin{bmatrix} C & -I & 0 & 0 & \cdots \\ -I & C & -I & 0 & \cdots \\ 0 & -I & C & -I & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \qquad C = \begin{bmatrix} 4 & -1 & 0 & 0 & \cdots \\ -1 & 4 & -1 & 0 & \cdots \\ 0 & -1 & 4 & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}.$$

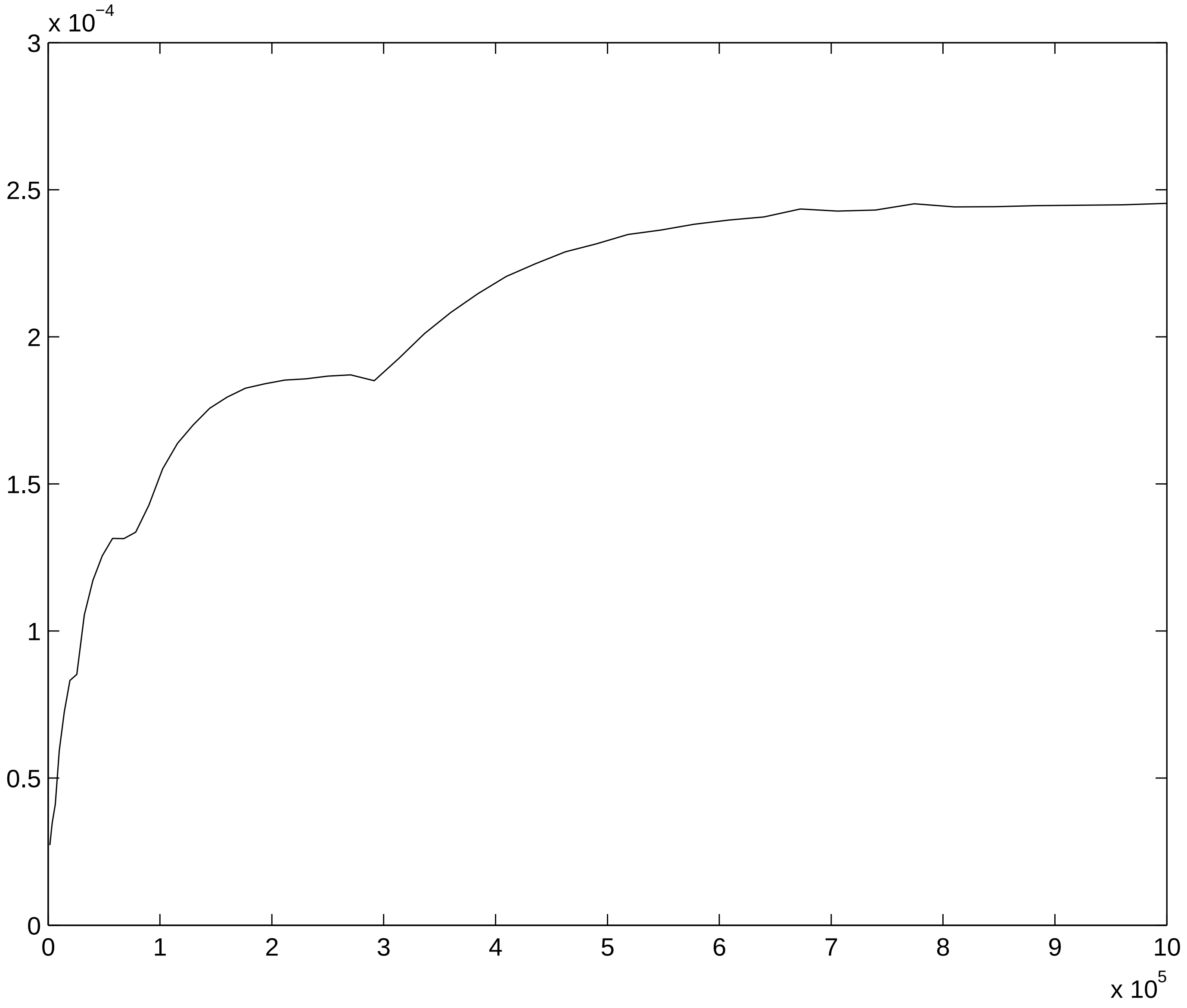
N	T_{invert}	T_{apply}	M	e_1	e_2	e_3	e_4
	(seconds)	(seconds)	(kB)				
10 000	5.93e-1	2.82e-3	3.82e+2	1.29e-8	1.37e-7	2.61e-8	3.31e-8
40 000	4.69e+0	6.25e-3	9.19e+2	9.35e-9	8.74e-8	4.71e-8	6.47e-8
90 000	1.28e+1	1.27e-2	1.51e+3	—	—	7.98e-8	1.25e-7
160 000	2.87e+1	1.38e-2	2.15e+3	—	—	9.02e-8	1.84e-7
250 000	4.67e+1	1.52e-2	2.80e+3	—	—	1.02e-7	1.14e-7
360 000	7.50e+1	2.62e-2	3.55e+3	—	—	1.37e-7	1.57e-7
490 000	1.13e+2	2.78e-2	4.22e+3	—	—	—	—
640 000	1.54e+2	2.92e-2	5.45e+3	—	—	—	—
810 000	1.98e+2	3.09e-2	5.86e+3	—	—	—	—
1 000 000	2.45e+2	3.25e-2	6.66e+3	—	—	—	—

e_1 The largest error in any entry of $\tilde{A}_n^{-1}$

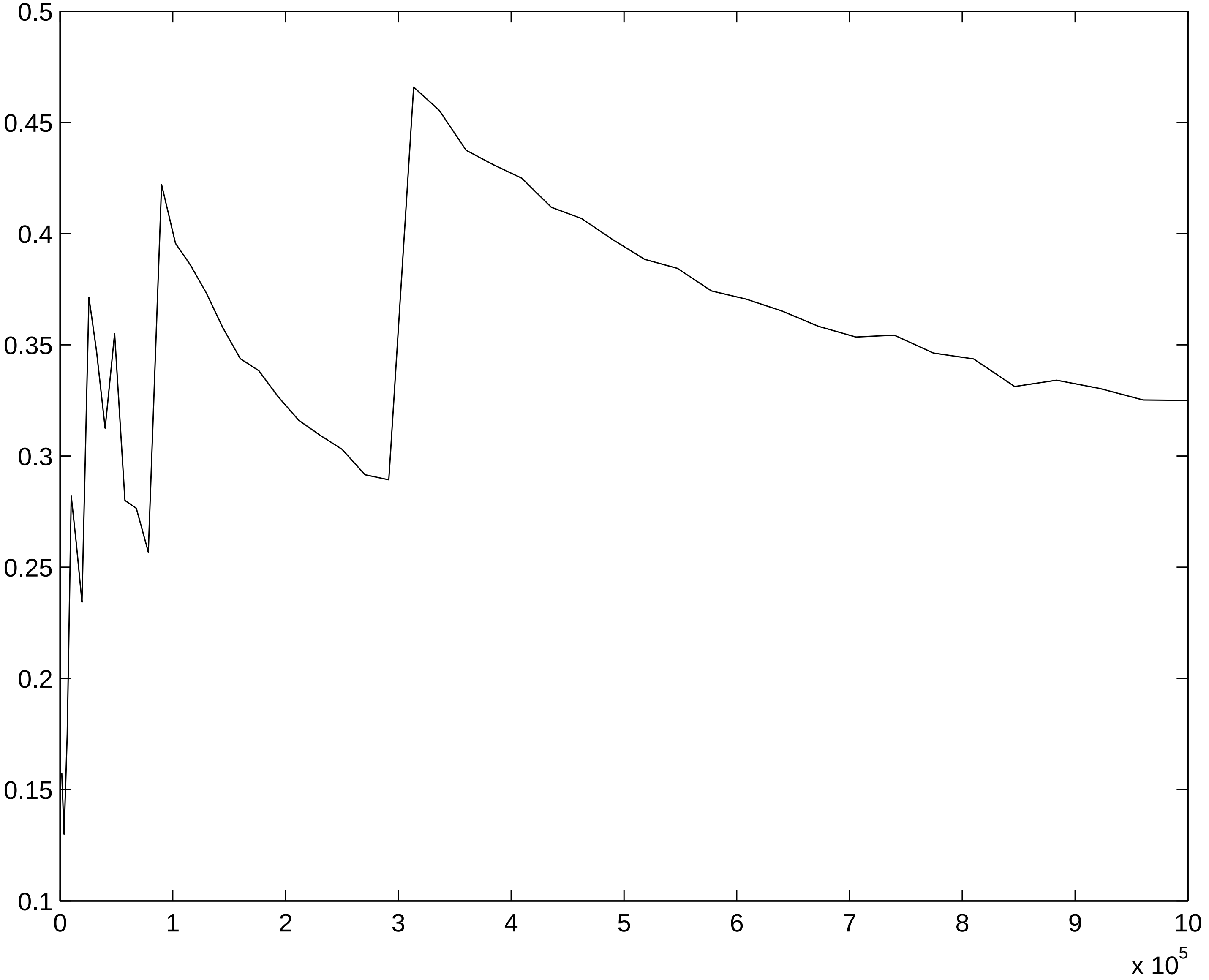
e_2 The error in l^2-operator norm of $\tilde{A}_n^{-1}$

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

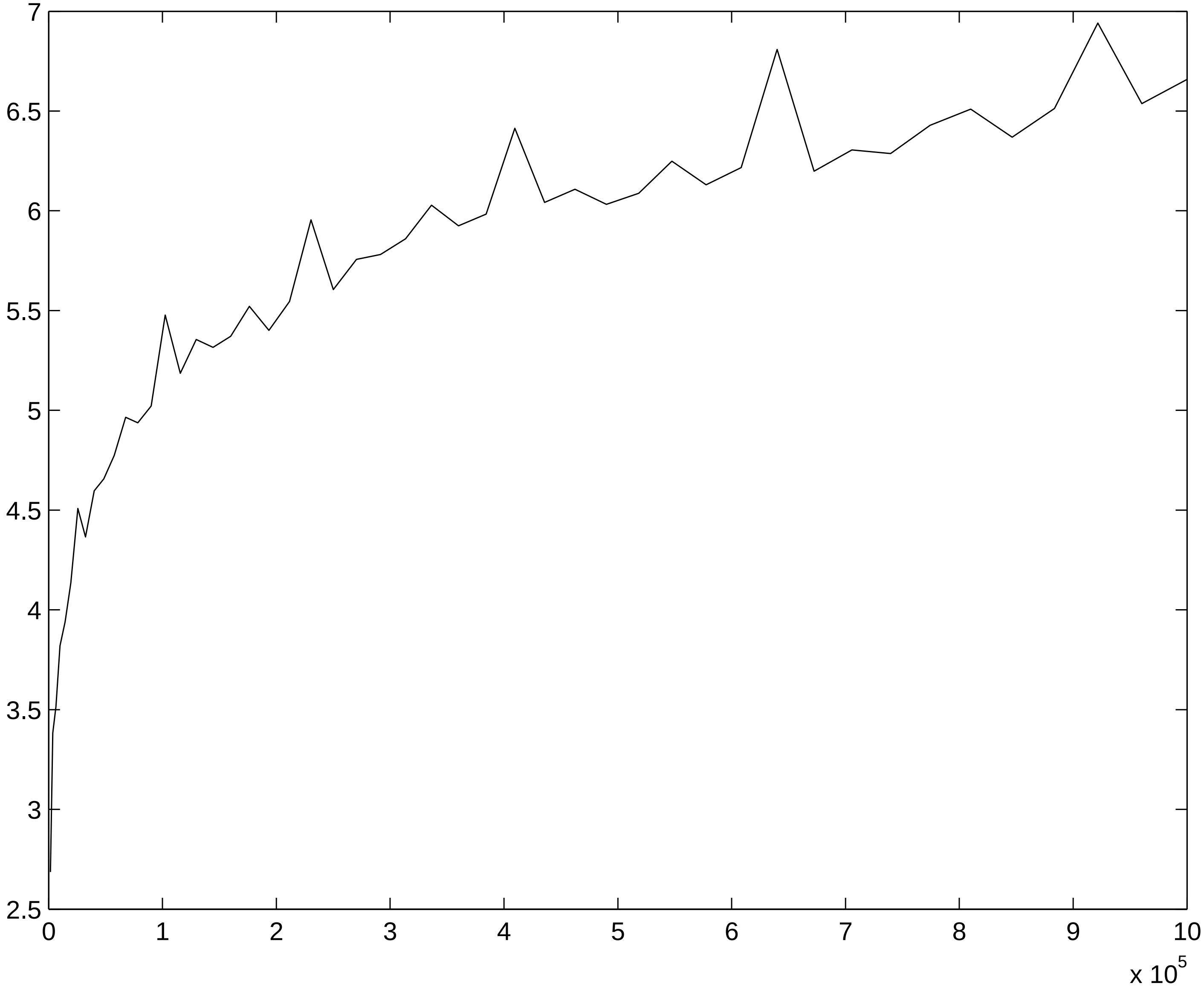
e_4 The l^2-error in the first column of $\tilde{A}_{nn}^{-1}$.



$$\frac{T_{\text{invert}}}{N}\ \text{versus}\quad N$$



$$\frac{T_{\text{apply}}}{\sqrt{N}} \quad \text{versus} \quad N$$



$$\frac{M}{\sqrt{N}} \text{ versus } N.$$

Main topic of talk — fast direct solvers

- 2D boundary integral equations. Finished. $O(N)$. Very fast.
  Has proven capable of solving previously intractable problems.
- 2D volume problems (finite element matrices and Lippmann-Schwinger).
  Theory finished. Some code exists. $O(N)$ or $O(N\log(N))$. Work in progress.
- 3D surface integral equations. $O(N\ \log(N))$. "In principle" done. (Or is it?)

Future directions:

- Extensions of fast direct solvers:
  - Spectral decompositions.
  - High-frequency problems. (High risk / high reward type project.)
- Efforts to make PDE solvers based on integral equations more accessible:
  - Improve machinery for dealing with surfaces.
  - Assemble software packages.
- Applications of newly developed PDE solvers.
- Extension of methodology to do coarse graining / model reduction:
  - Composite media, crack propagation, percolating micro-structures.
  - Scattering problems.