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.

Why construct numerical methods for linear PDEs?

Well, ...

Seriously, isn't it known how to do this already?!?

Not in all environments, in particular when it comes to oscillatory problems (high frequency scattering problems, *etc*).

More importantly, this is one of the most commonly occurring computational tasks in scientific computing. *Significant* improvements in speed, accuracy, and robustness would have transformative effects on computational science.

For instance, as far as linear elliptic boundary value problems go (Laplace, elasticity, *etc*), the goal is instantaneous solves at ten digits of accuracy or more.

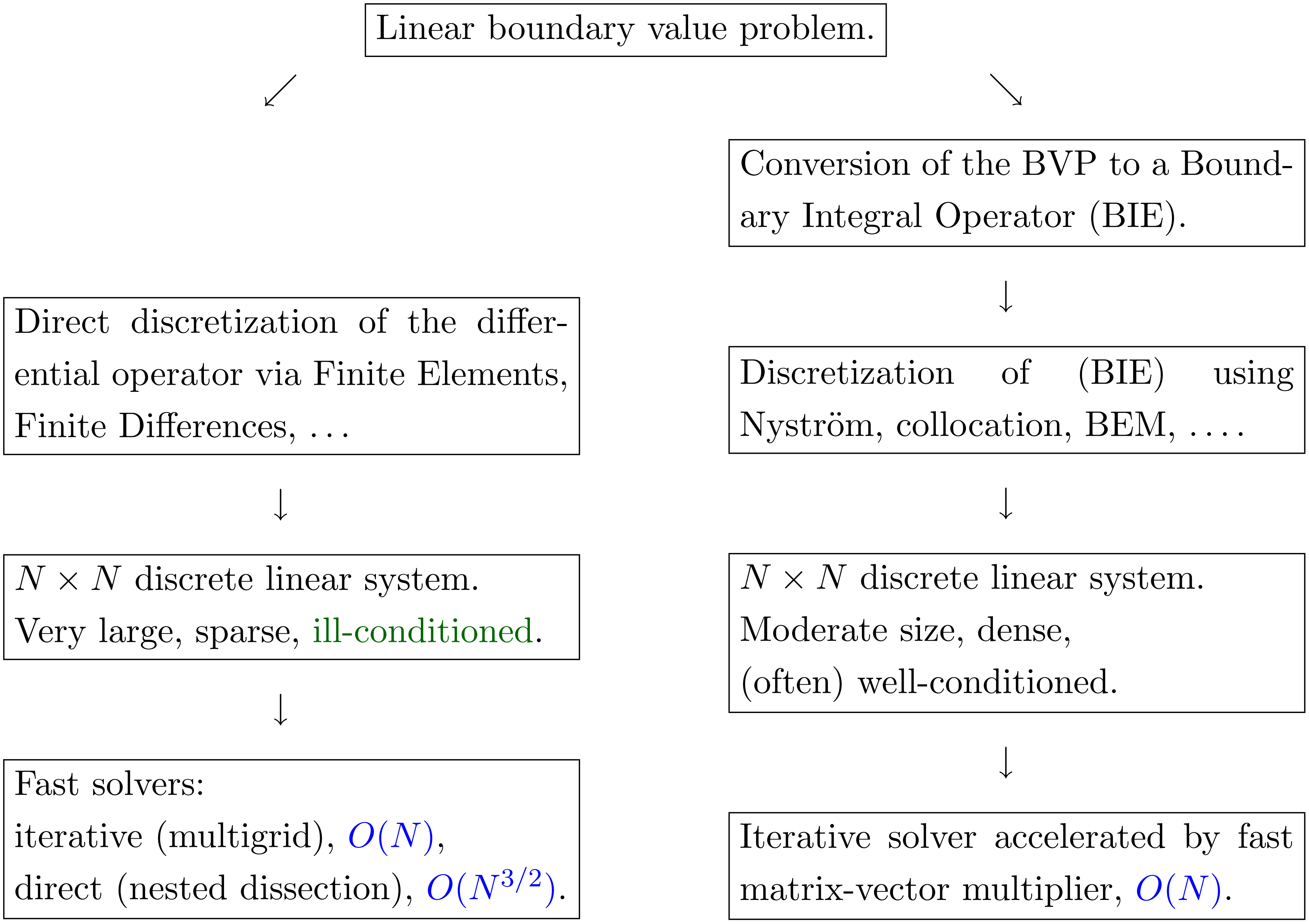
The challenge is similar to the task of constructing faster computers.

Outline of talk:

1: Linear PDE solvers — background, context.

2: $O(N)$ direct solvers.

3: Randomized sampling for constructing low-rank approximations to operators.



Linear boundary value problem.

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

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

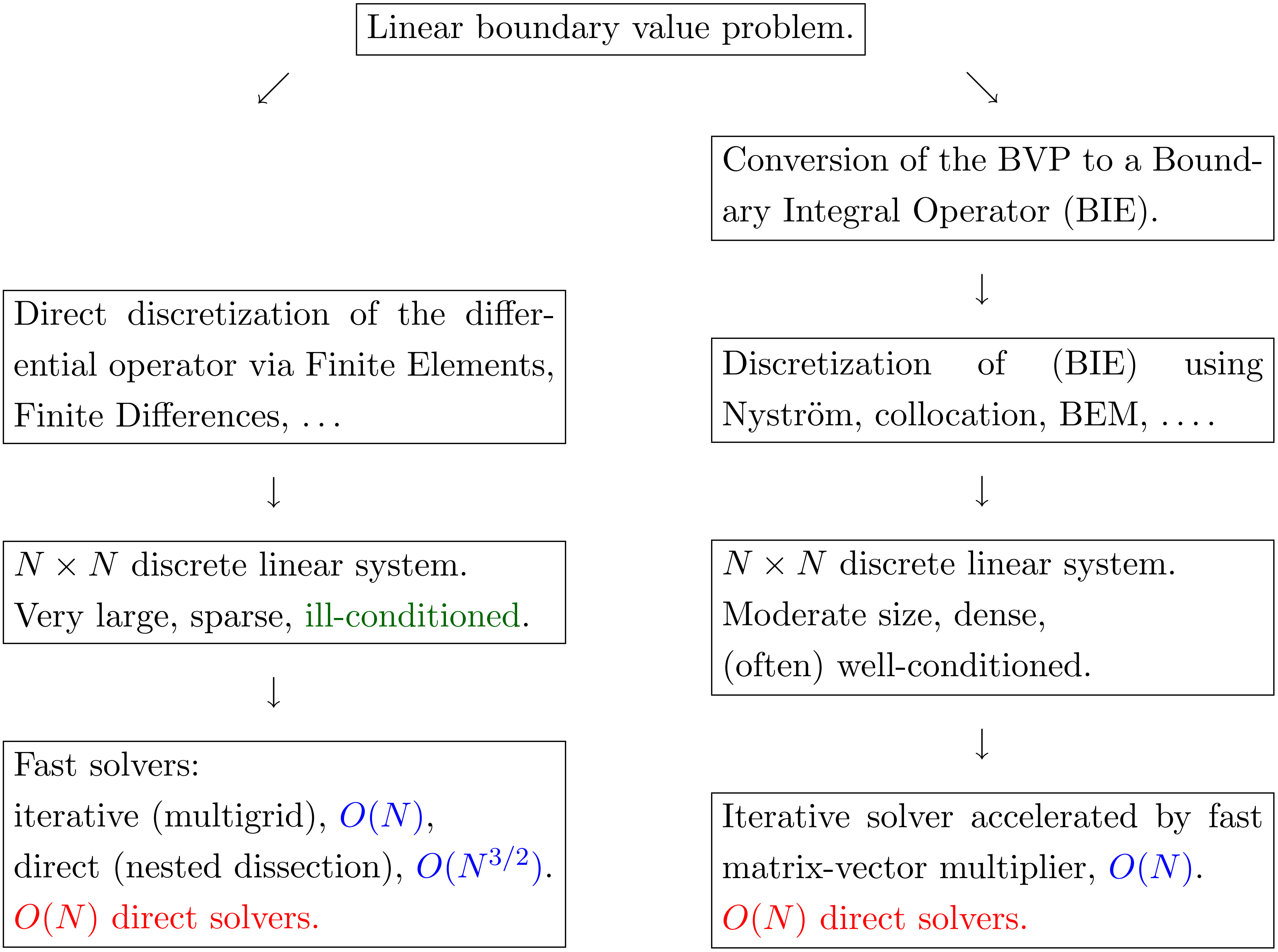
Fast solvers:
iterative (multigrid), $O(N)$,
direct (nested dissection), $O(N^{3/2})$.

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, $O(N)$.



Linear boundary value problem.

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

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

- The kernel k is derived from the operator A.

- The data function h is derived from the data of (BVP).

- The conversion from (BVP) to (BIE) sometimes involves the evaluation of certain integrals over Γ and/or Ω.

- Sometimes the integral equation must be formulated on Ω (*e.g.* for problems with low-order terms that have variable coeffiecients).

- ...

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 (2^{nd} 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)
$$(I + K_N)\, u_N = f_N$$

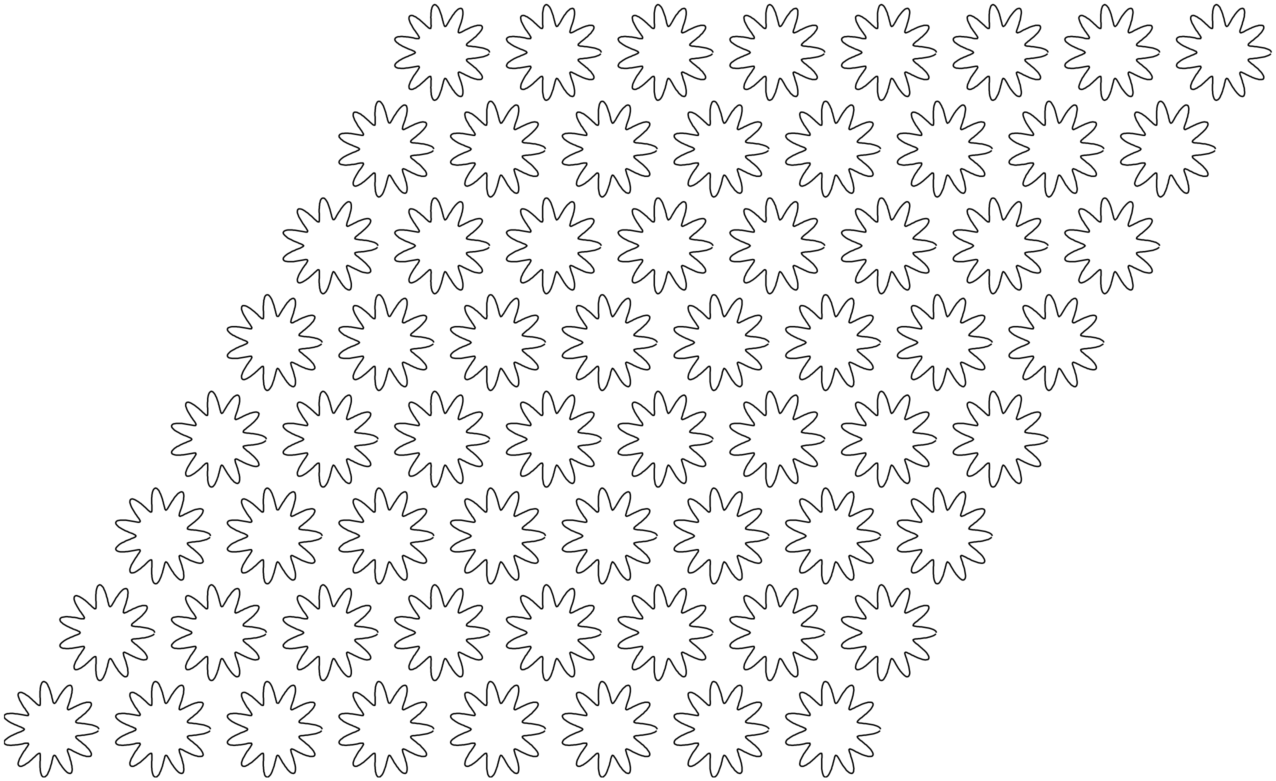
where K_N is a (typically dense) $N \times N$ matrix.

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 existing iterative ones.

WHAT IS A DIRECT SOLVER?

Recall that many BVPs can be cast in the following form:

(BIE) $$u(x) + \int_\Gamma g(x,y)u(y)\,ds(y) = f(x), \qquad x \in \Gamma.$$

Upon discretization, equation (BIE) turns into a discrete equation

(DISC) $$(I + K_N)u = f$$

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

A ***direct method*** computes a compressed representation for $(I + K_N)^{-1}$.

- Cost for pre-computing the inverse.
- Cost for applying the inverse to a vector.

In many environments, both of these costs can be made $O(N)$.

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. (Yes, yes, yes,. . .)

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.*

How does the inversion scheme work?

By exploiting rank deficiencies in the off-diagonal blocks.

Note: Problems with highly oscillatory kernels such as high-frequency Helmholtz can currently not be handled. (However, things work great at low and intermediate frequencies.)

The scheme is a multi-level algorithm operating on a hierarchical partitioning of the computational domain.

In progressing from one level to the next coarser one, functions are split into a fine scale part and a coarse scale part and the fine scale part is eliminated.

One-level compression:

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

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

We set

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

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

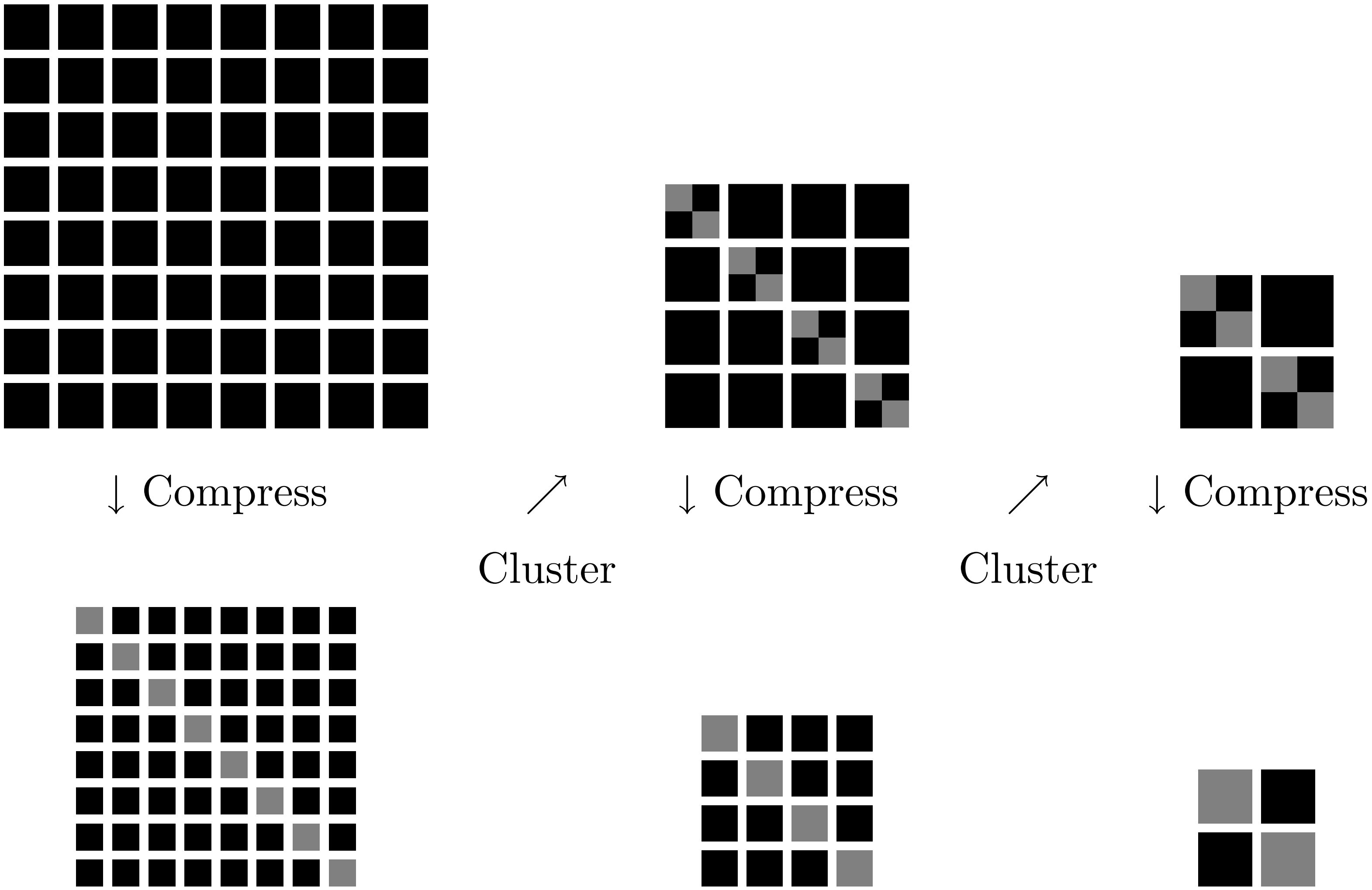
$$\begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} & \tilde{A}_{13} & \tilde{A}_{14} \\ \tilde{A}_{21} & \tilde{A}_{22} & \tilde{A}_{23} & \tilde{A}_{24} \\ \tilde{A}_{31} & \tilde{A}_{32} & \tilde{A}_{33} & \tilde{A}_{34} \\ \tilde{A}_{41} & \tilde{A}_{42} & \tilde{A}_{43} & \tilde{A}_{44} \end{bmatrix}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

where

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

(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

The critical step is to find matrices U_j such that when $i \neq j$,

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

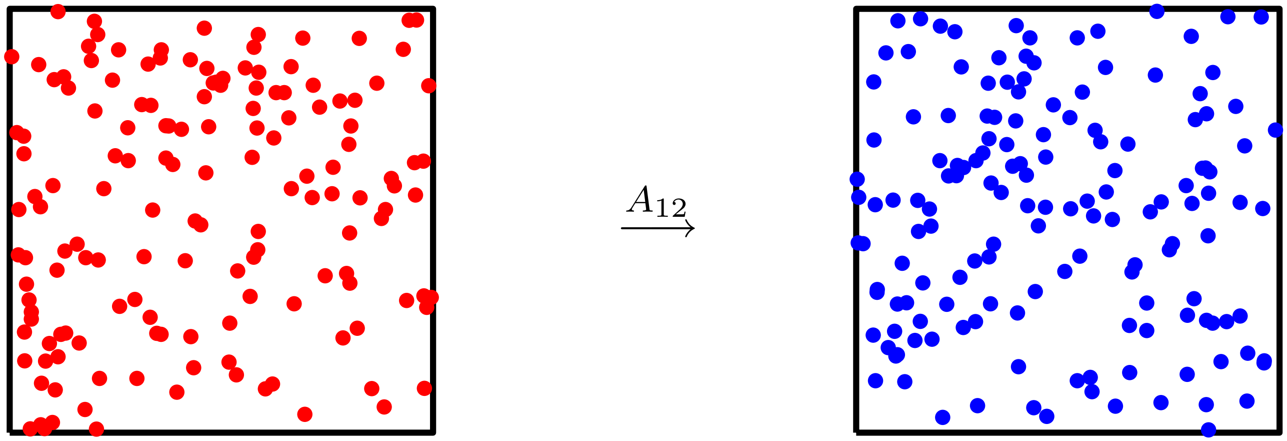
for some matrix $\tilde{A}_{ij}$ that is smaller than A_{ij}.

To attain an $O(N)$ scheme, *one cannot afford to even look at every off-diagonal block*. Instead, one can use:

- 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 that the kernel must satisfy [Martinsson, Rokhlin].
  - Very robust.
  - Leads to representations that are very close to optimal.
- Randomized sampling. New!

To further improve the operation counts, we use factorizations such that
$\tilde{A}_{ij}$ is a submatrix of A_{ij} when $i \neq j$.

The concept of “proxy charges”:



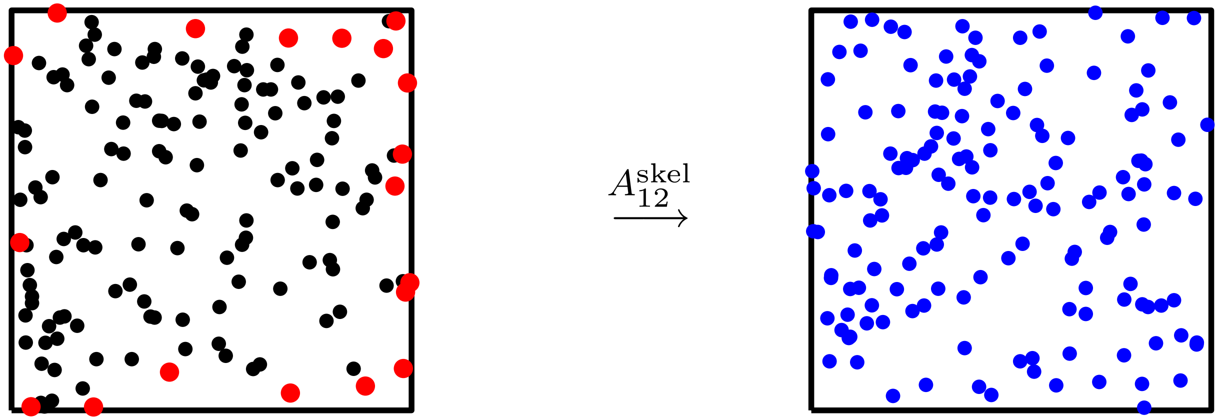
Sources $\{q_n\}_{n=1}^N$

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

$$\begin{tikzcd}
\{q_n\}_{n=1}^N \arrow[r, "A_{12}"] \arrow[d, "?"'] & \{v_m\}_{m=1}^M \\
\text{“Small” representation} \arrow[ur, "?"'] &
\end{tikzcd}$$

The key observation is that $k = \mathrm{rank}(A_{12}) < \min(M, N)$.

Skeletonization

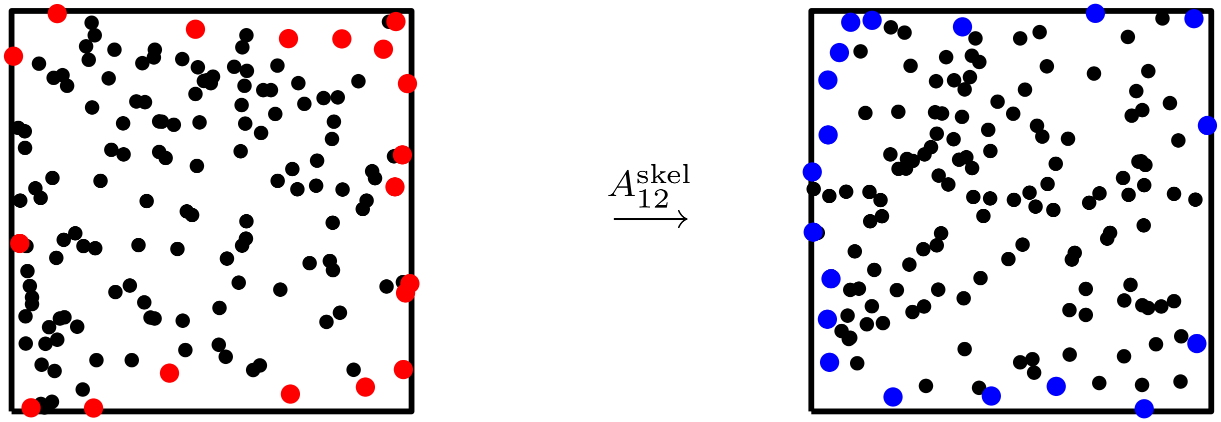


$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[r, "A_{12}"] \arrow[d, "U_2^{\mathrm{t}}"'] & \{v_m\}_{m=1}^{M} \\
\{\tilde{q}_{n_j}\}_{j=1}^{k} \arrow[ur, "A_{12}^{\mathrm{skel}}"'] &
\end{tikzcd}$$

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_{12}^{skel} is a submatrix of A_{12}.

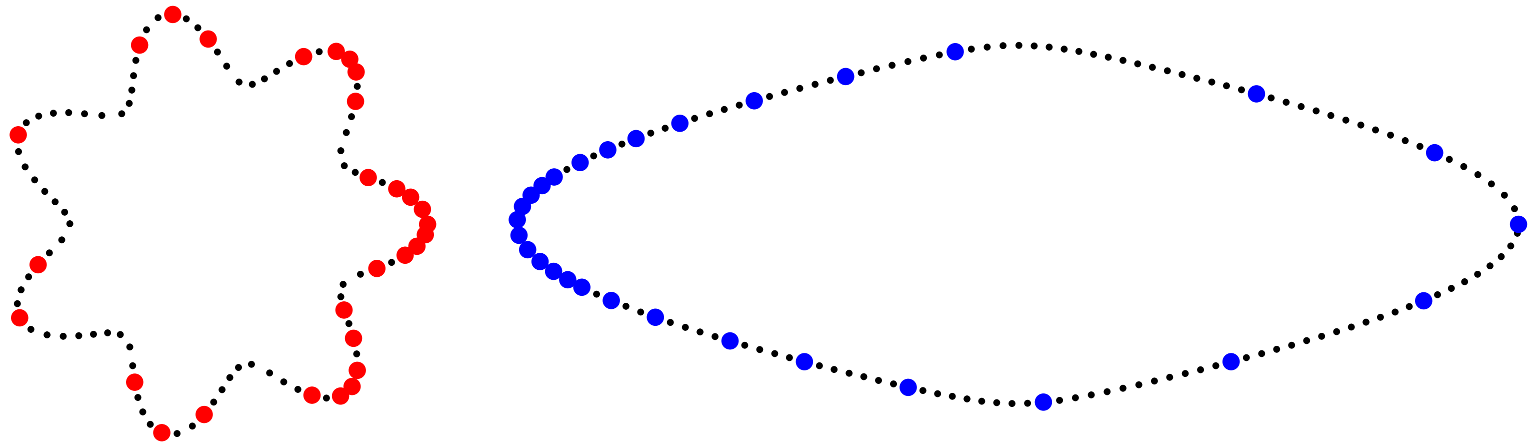
We can “skeletonize” both Ω_1 and Ω_2.



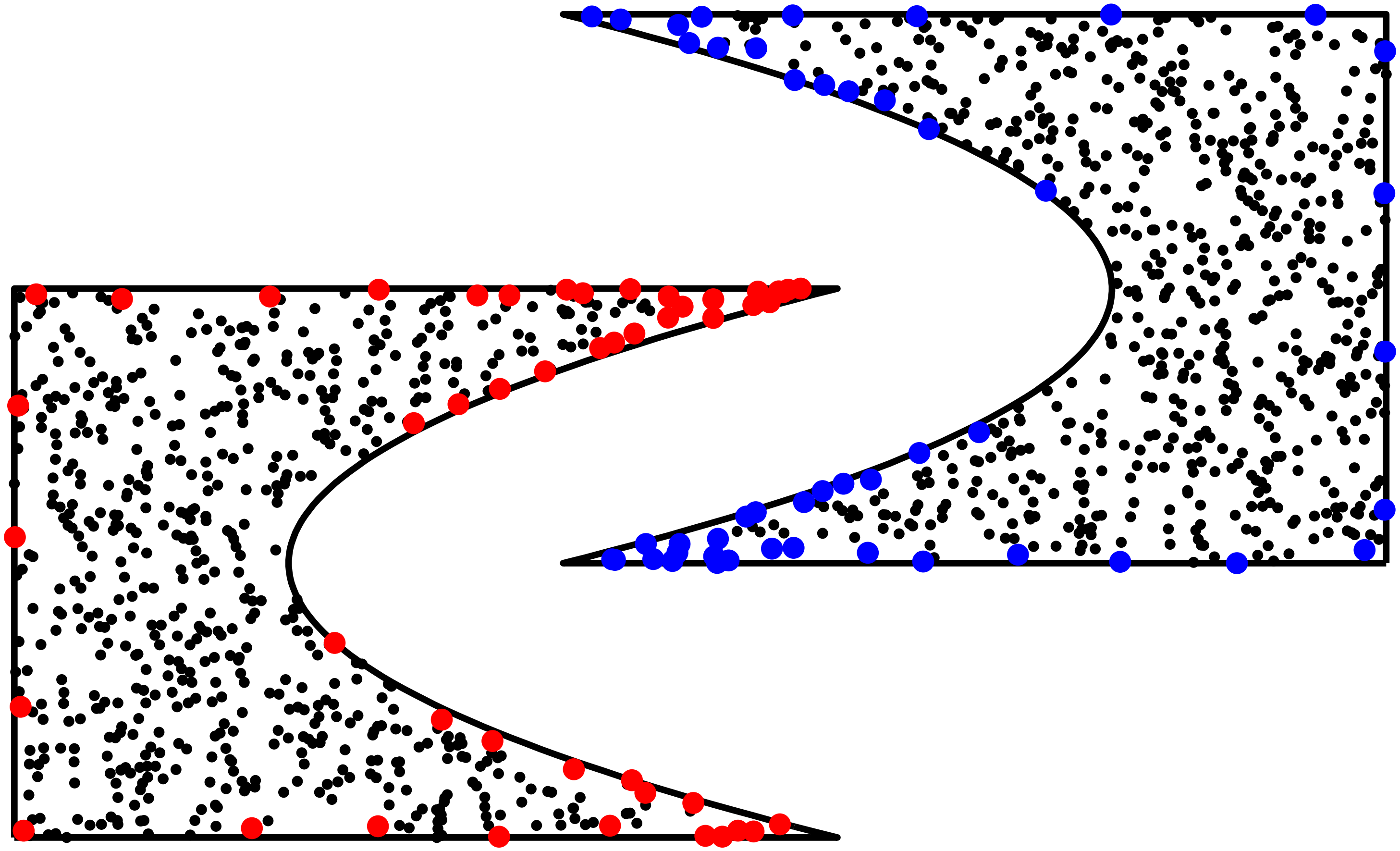
$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[r, "A_{12}"] \arrow[d, "U_2^{\text{t}}"'] & \{v_m\}_{m=1}^{M} \\
\{\tilde{q}_{n_j}\}_{j=1}^{k} \arrow[r, "A_{12}^{\text{skel}}"'] & \{v_{m_j}\}_{j=1}^{k} \arrow[u, "U_1"']
\end{tikzcd}$$

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

Skeletonization can be performed for Ω_{S} and Ω_{T} of various shapes.

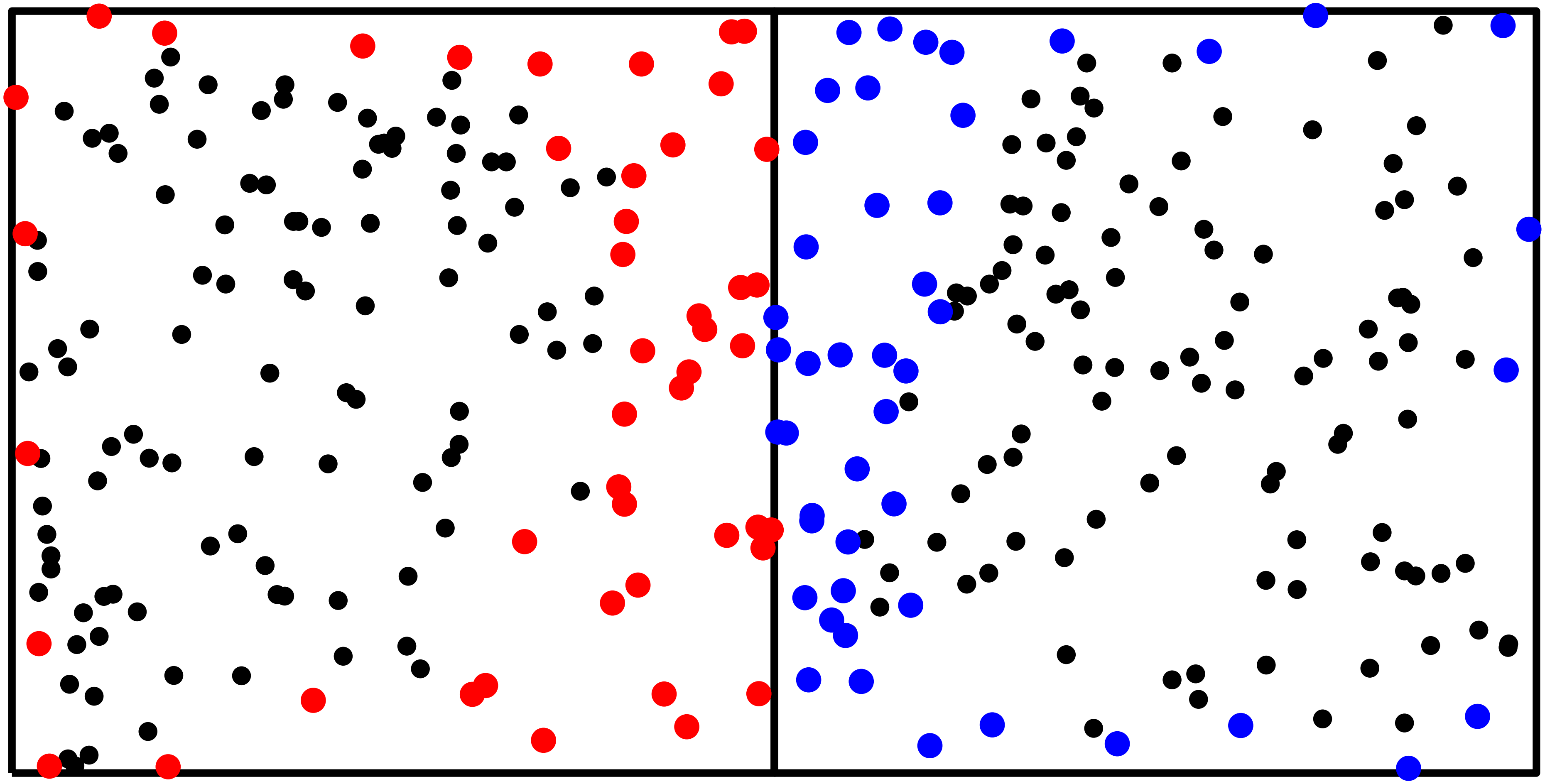


Rank $= 29$ at $\varepsilon = 10^{-10}$.



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

Adjacent boxes can be skeletonized.



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

$$\begin{tikzcd}
\{q_n\}_{n=1}^{N} \arrow[r, "A_{12}"] \arrow[d, "U_2^{\mathrm{t}}"'] & \{v_m\}_{m=1}^{M} \\
\{\tilde{q}_{n_j}\}_{j=1}^{k} \arrow[r, "A_{12}^{\mathrm{skel}}"'] & \{v_{m_j}\}_{j=1}^{k} \arrow[u, "U_1"']
\end{tikzcd}$$

Benefits:

- The rank is optimal.
- The projection and interpolation are cheap.
  U_1 and U_2 contain $k \times k$ identity matrices.
- The projection and interpolation are well-conditioned.
- Finding the k points is cheap.
- The matrix $\tilde{A}_{12}$ is a submatrix of the original matrix A_{12}.
  (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

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

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

Coarse resolution.
Small blocks.

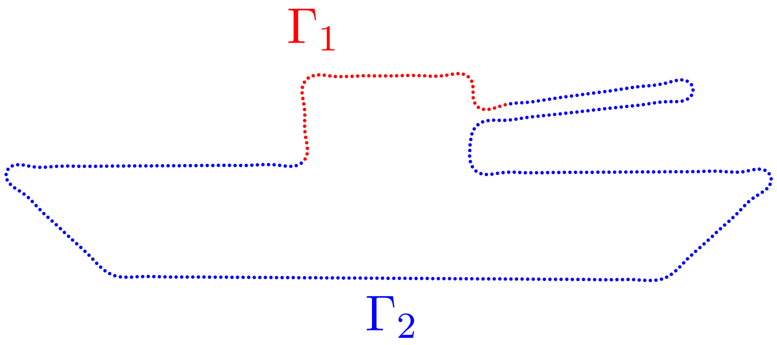
where A^{skel}_{ij} is a submatrix of A_{ij} when $i \neq j$.

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.

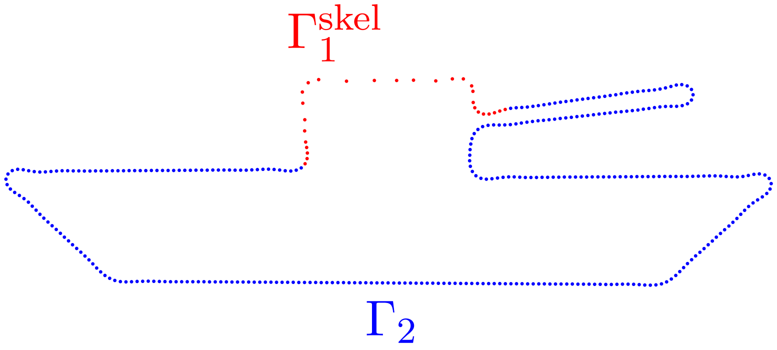
Let Γ_2 denote the rest of the domain.

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

$\tilde{A}_{11}$ 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.

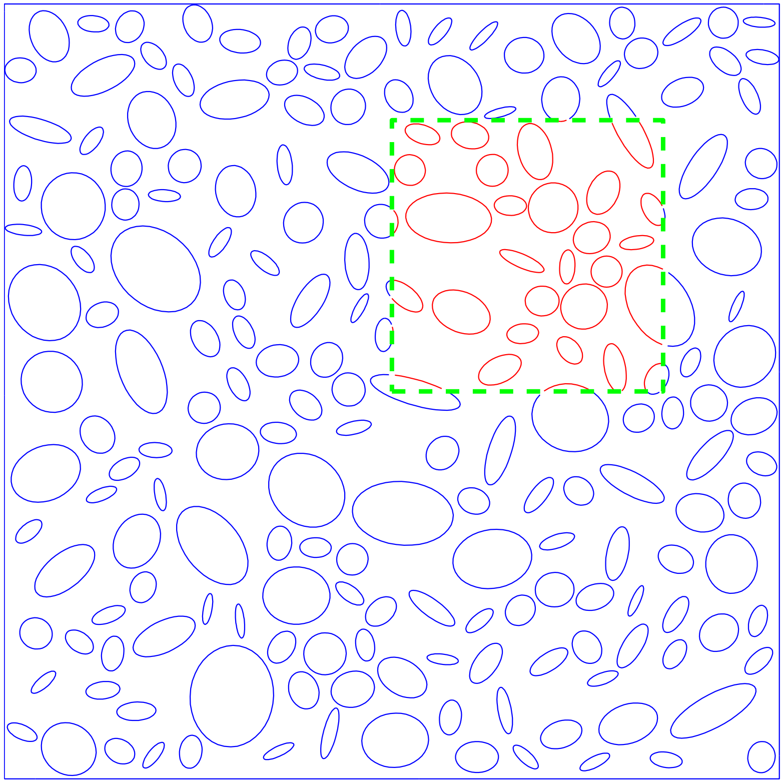
Let Γ_2 denote the rest of the domain.

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

$\tilde{A}_{11}$ 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

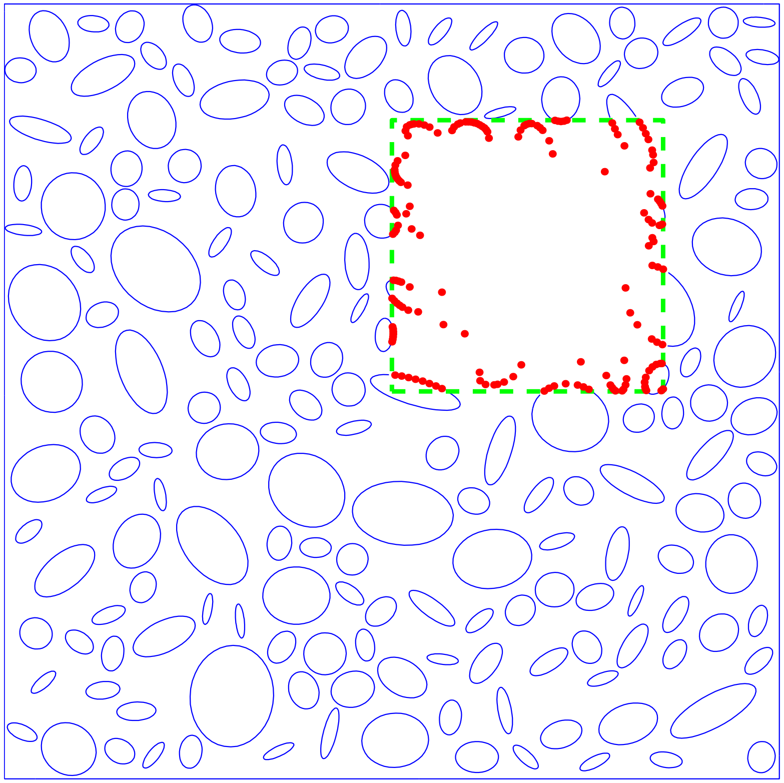
$$\underbrace{\tilde{A}_{ii}}_{k\times k} = \big(\underbrace{U_i^{\mathrm{t}}}_{k\times n}\,\underbrace{A_{ii}^{-1}}_{n\times n}\,\underbrace{U_i}_{n\times k}\big)^{-1}.$$

We call these blocks "proxy matrices".

What are they?

Let Ω_1 denote the block marked in red.

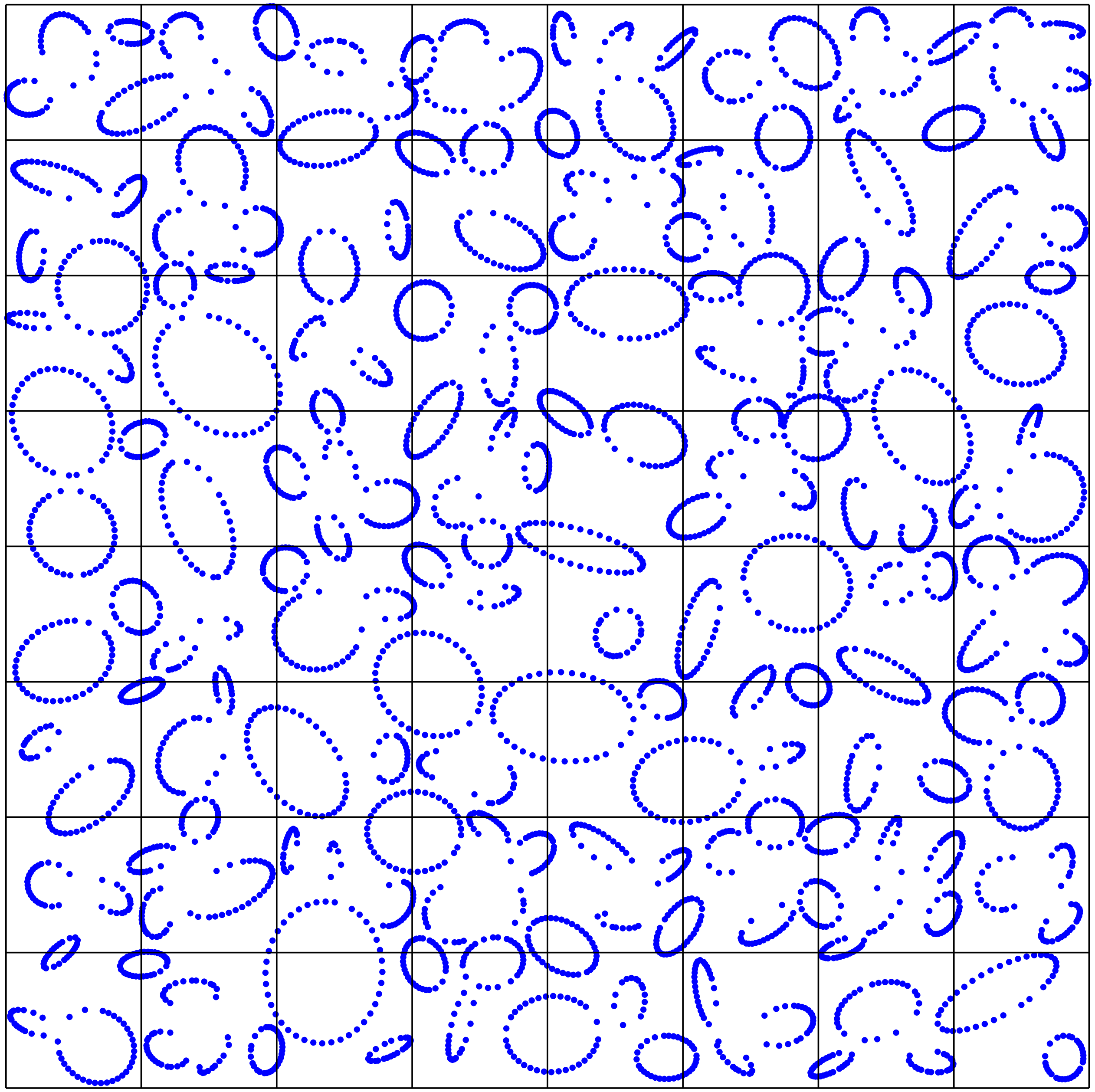
Let Ω_2 denote the rest of the domain.

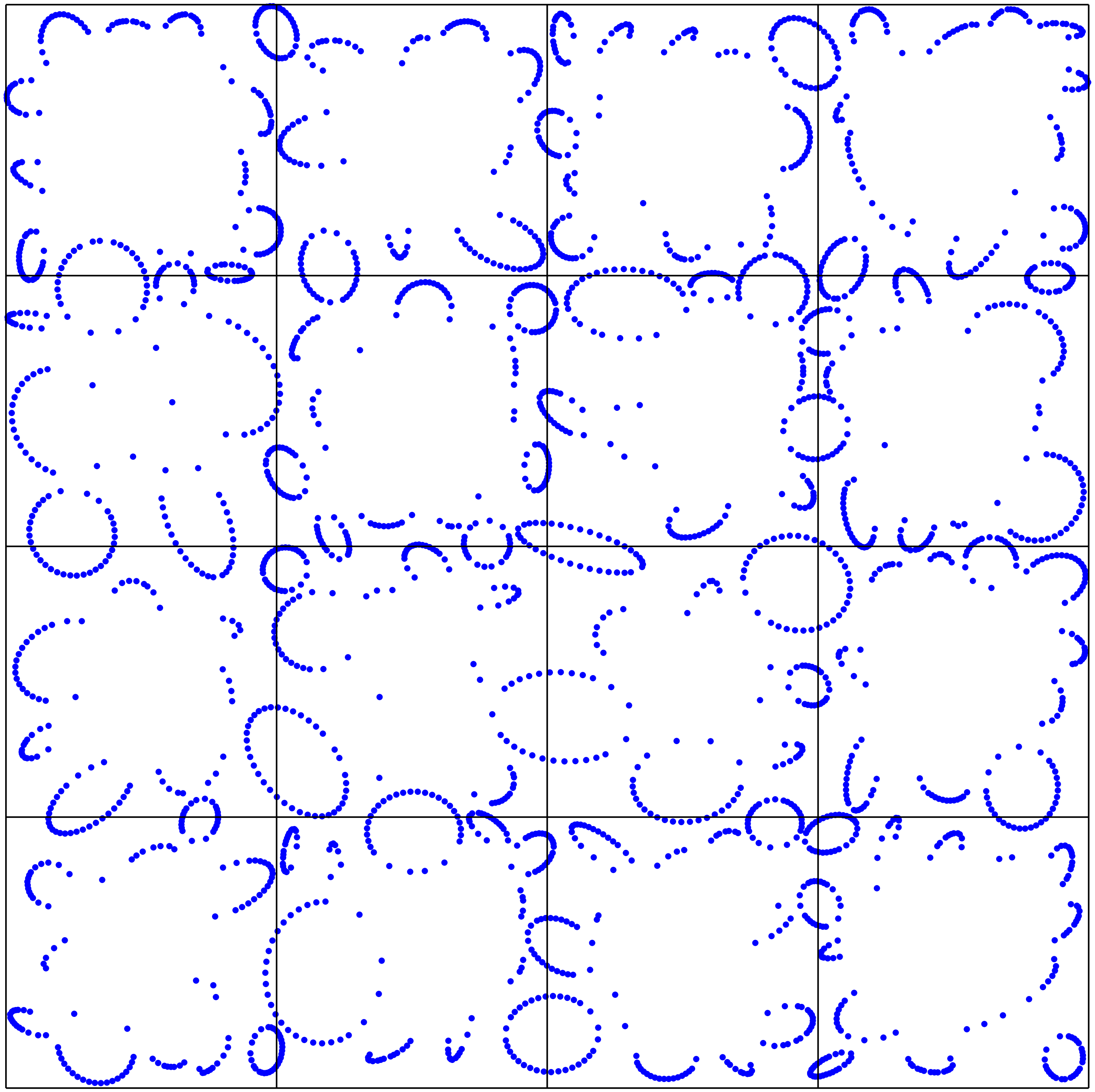


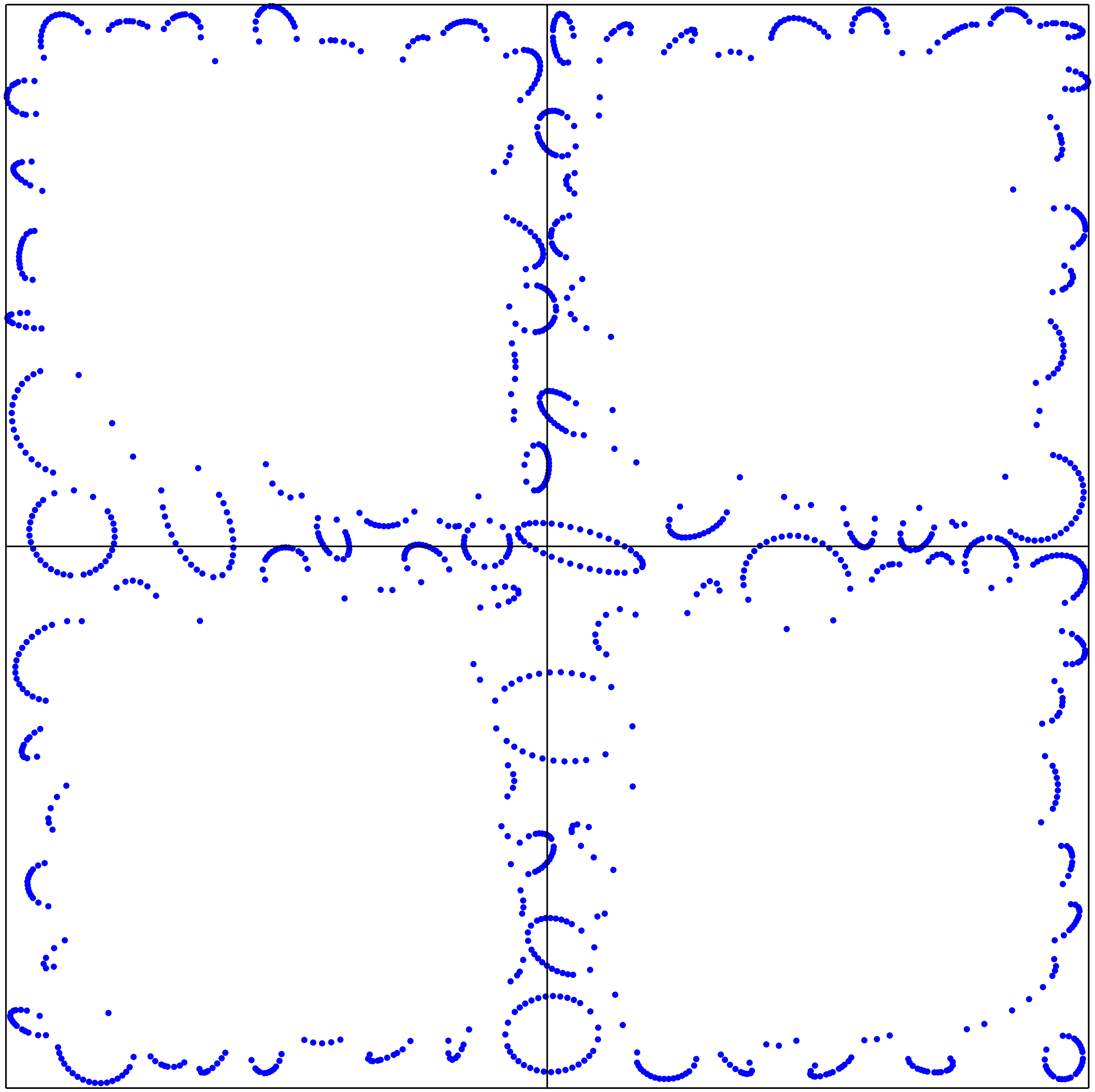
$$\begin{tikzcd}
\text{Charges on } \Omega_2 \arrow[r, "A_{12}"] \arrow[dr, "A_{12}^{\mathrm{skel}}"'] & \text{Pot. on } \Omega_1 \arrow[r, "A_{11}^{-1}"] & \text{Charges on } \Omega_1 \arrow[r, "A_{21}"] \arrow[d, "U_1^{\mathrm{t}}"'] & \text{Pot. on } \Omega_2 \\
 & \text{Pot. on } \Omega_1^{\mathrm{skel}}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

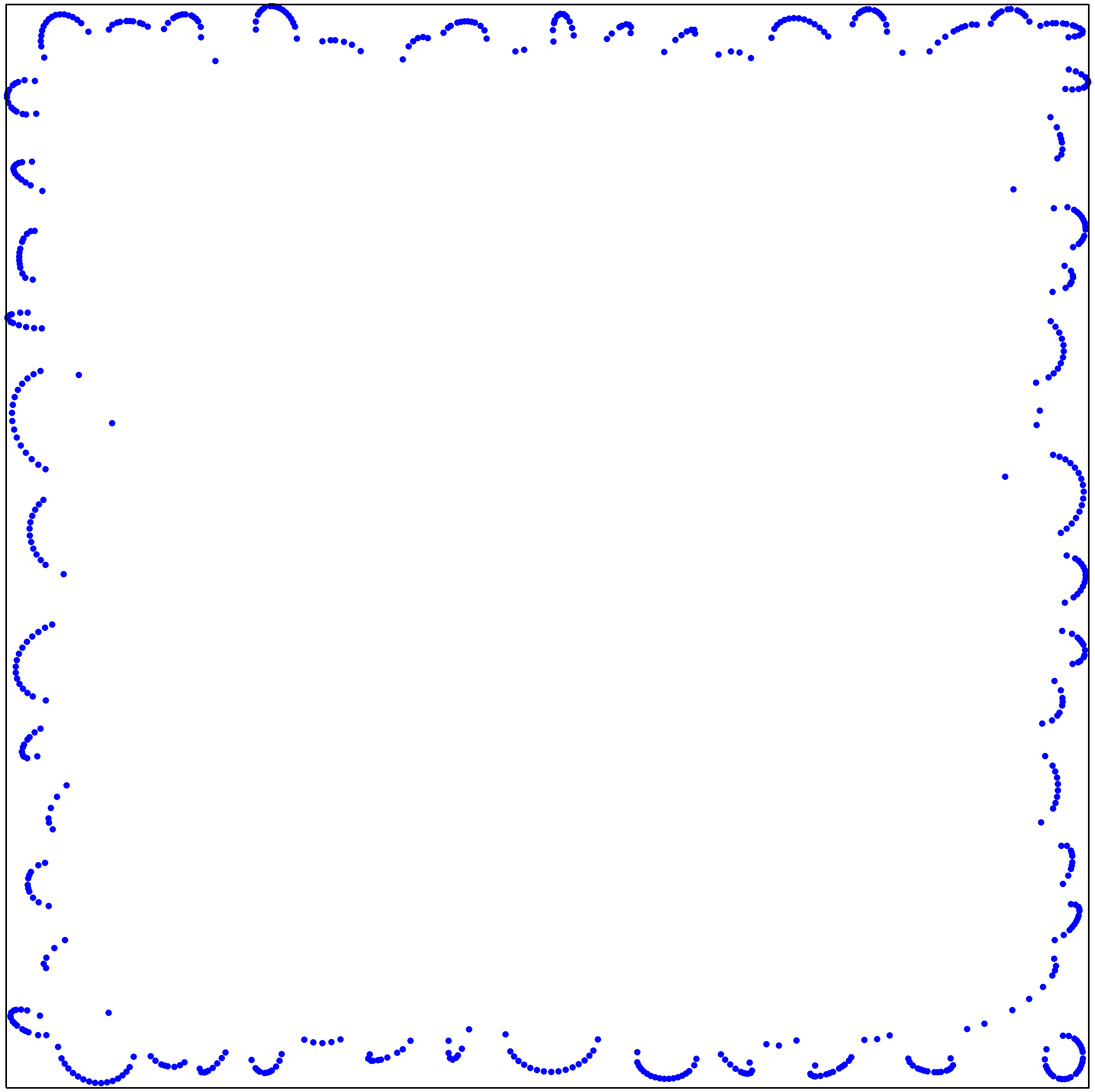
$\tilde{A}_{11}$ contains *all the information the outside world needs to know about* Ω_1.

To obtain a globally $O(N)$ scheme, we hierarchically merge proxy matrices.







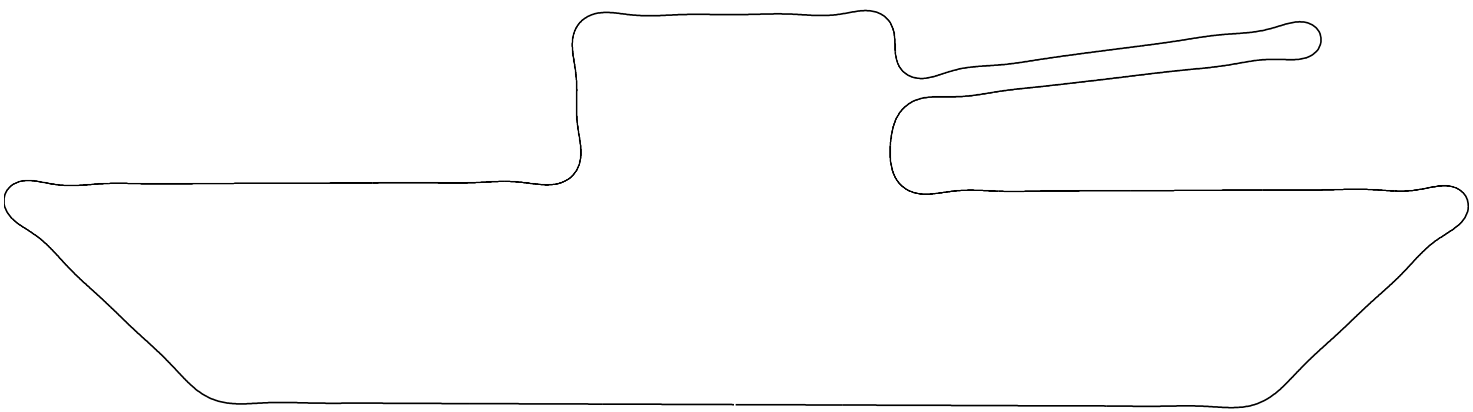


Numerical examples

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



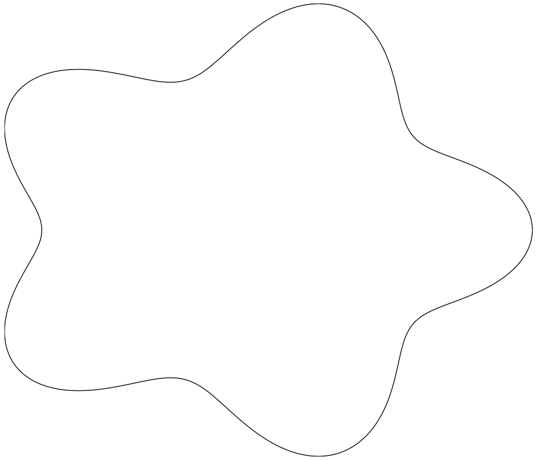
A smooth contour. Its length is roughly 15 and its horizontal width is 2.

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 10^{th} 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

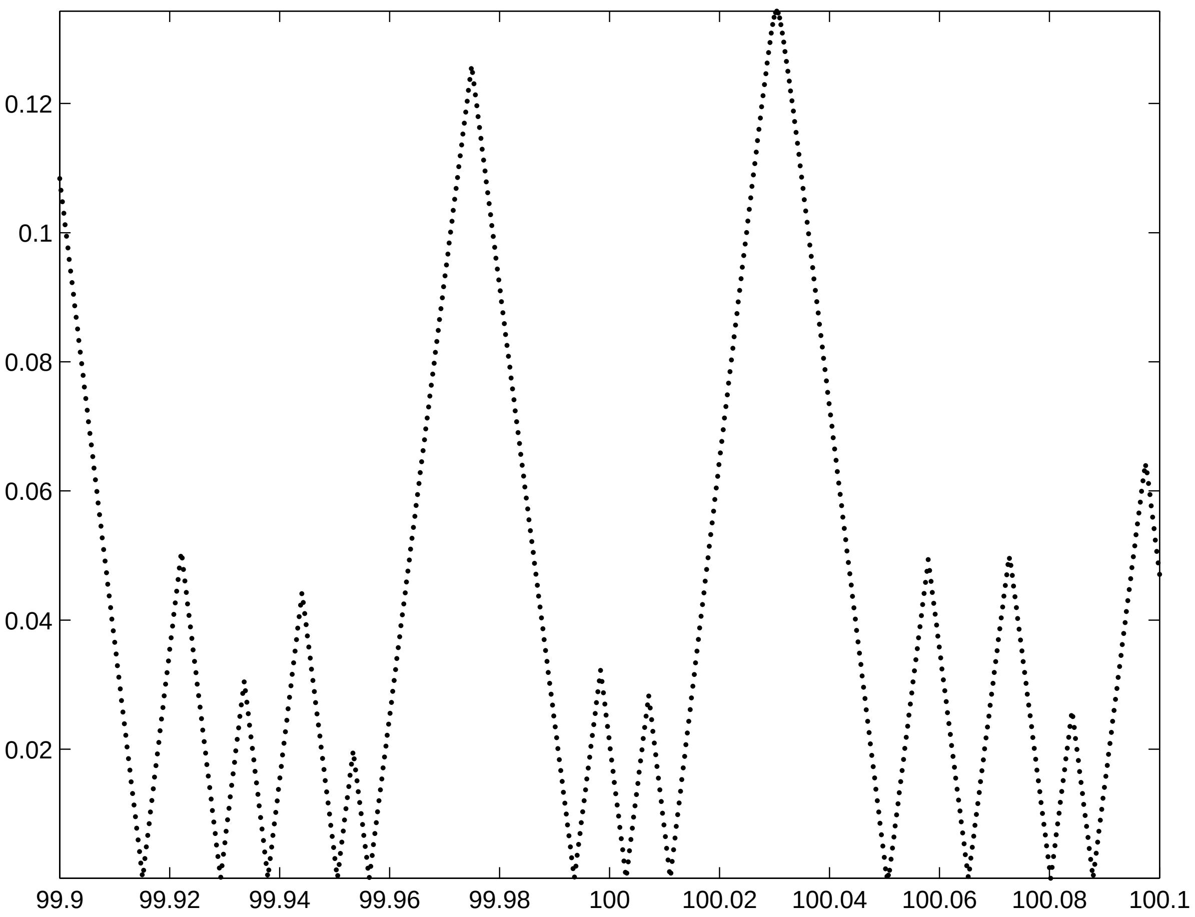


The diameter of the contour is about 2.5. An interior Helmholtz problem with Dirichlet boundary data was solved using $N = 6\,400$ discretization points, with a prescribed accuracy of 10^{-10}.

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

Time for constructing the inverse: 0.7 seconds.

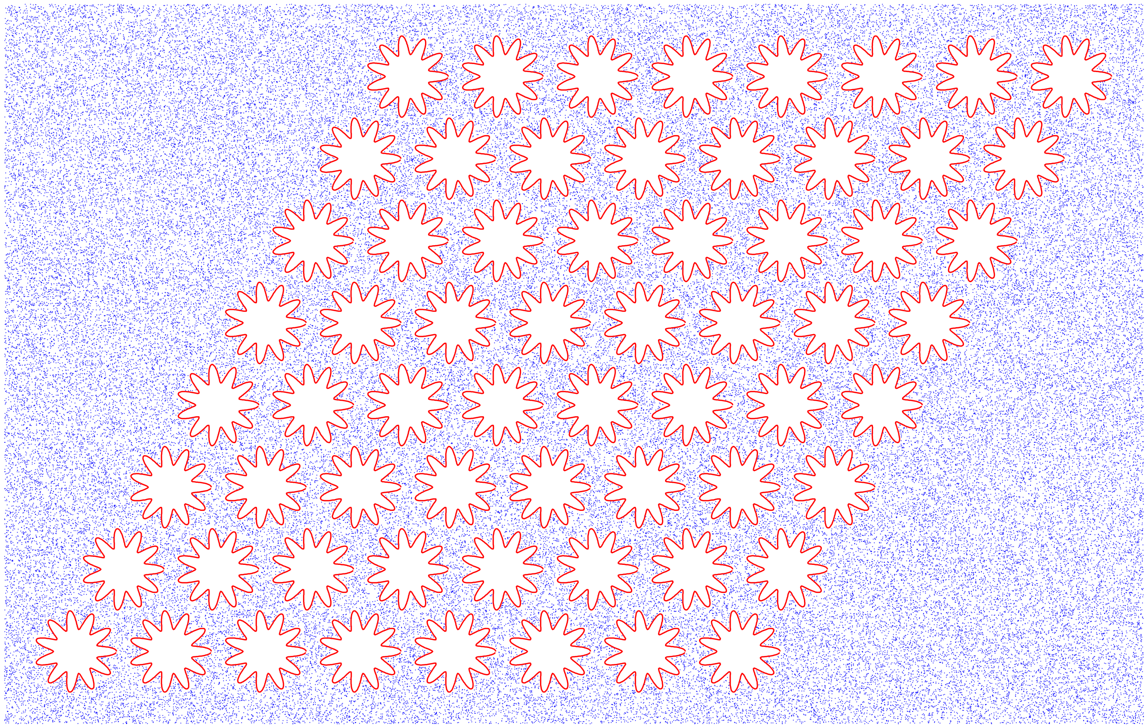
Error in the inverse: 10^{-5}.



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

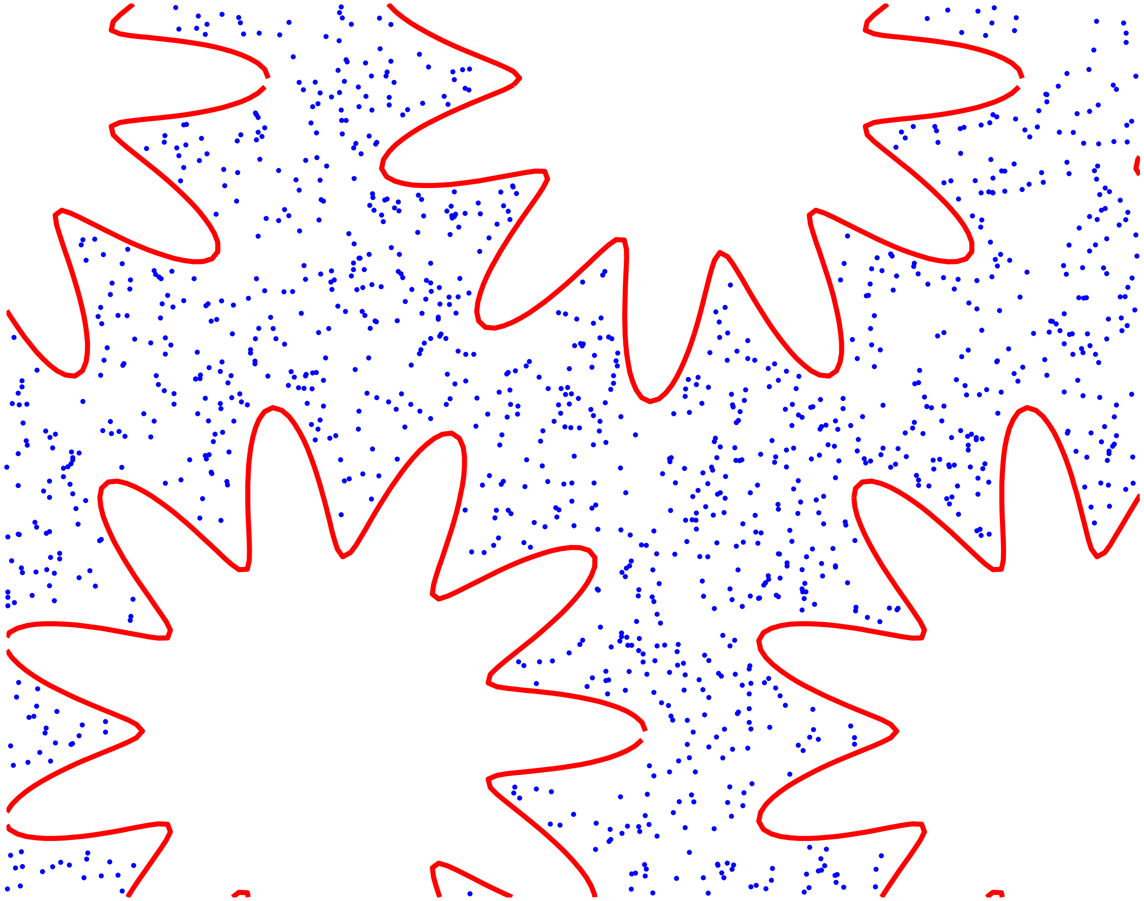


$\varepsilon = 10^{-5}$ $\quad N_{\text{contour}} = 25\,600$ $\quad N_{\text{particles}} = 100\,000$

Time to invert the boundary integral equation = 46sec.

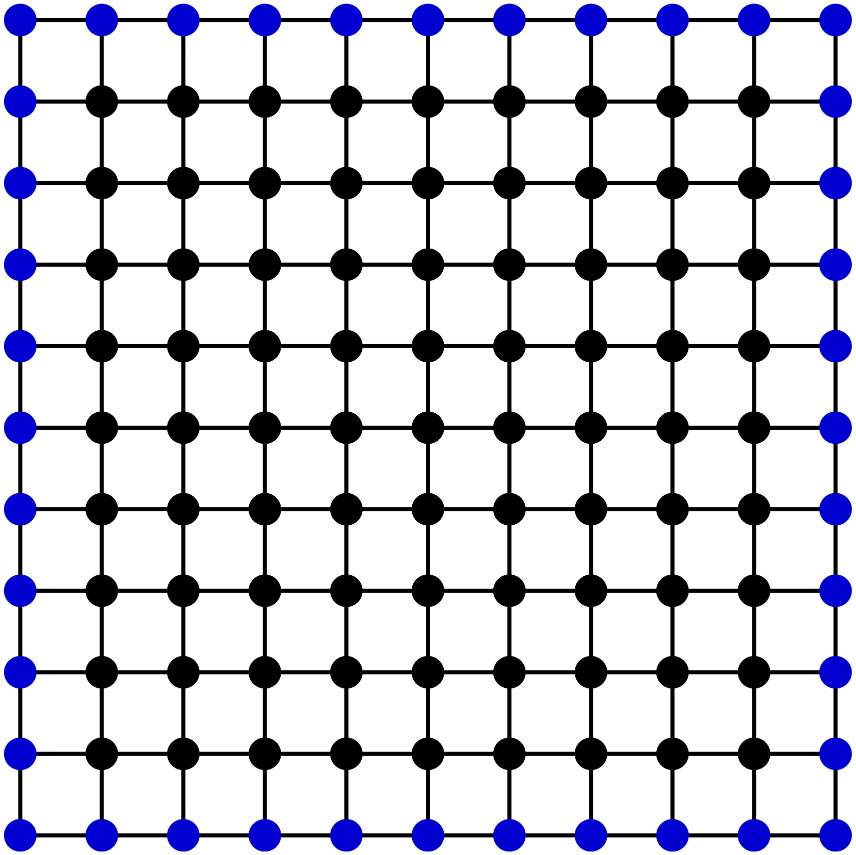
Time to compute the induced charges = 0.42sec.(2.5sec for the FMM)

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:

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

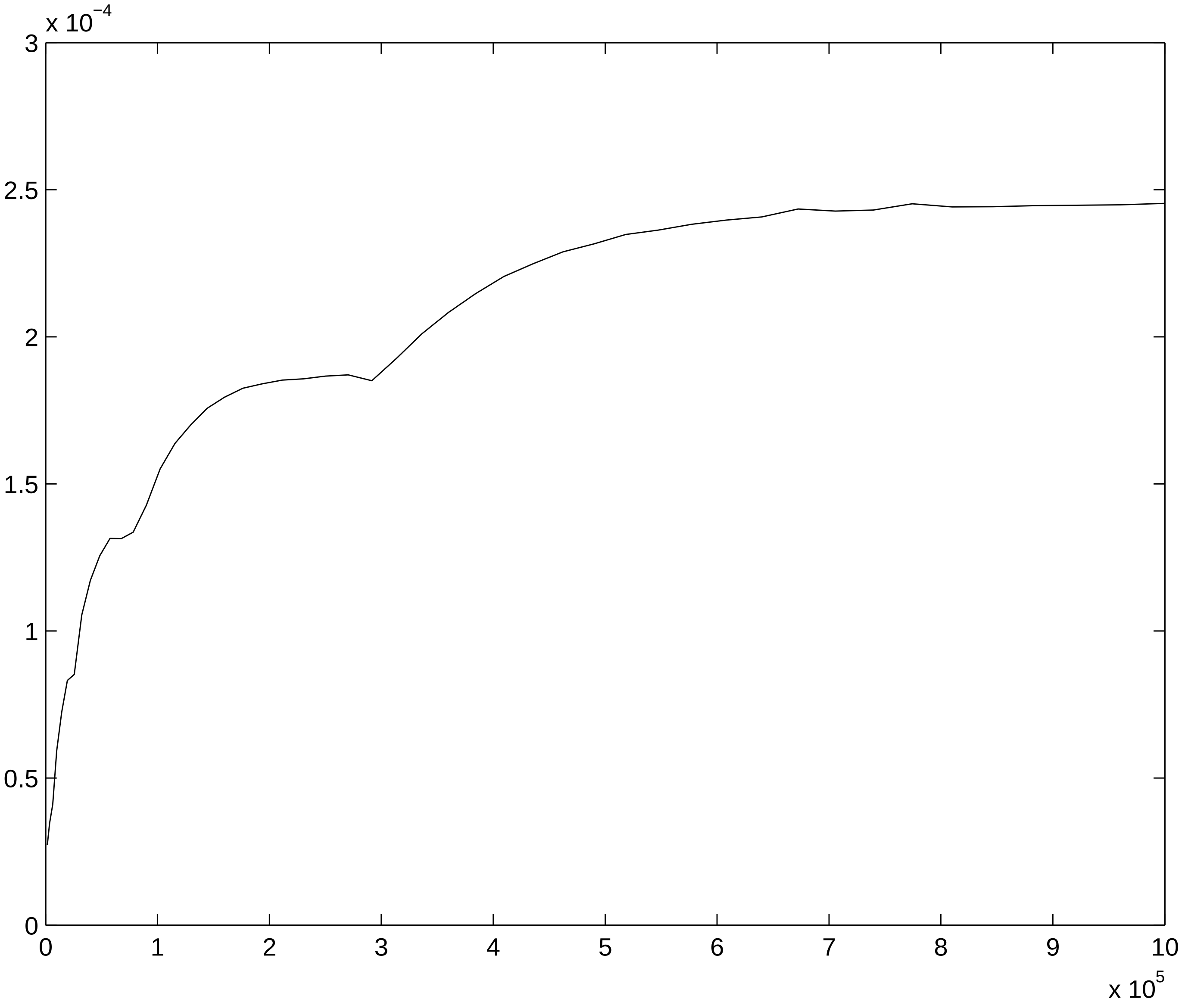
N	T_{invert}	T_{apply}	M	e_1	e_2	e_3	e_4
	(seconds)	(seconds)	(kB)				
10 000	5.93e-1	2.82e-3	3.82e+2	1.29e-8	1.37e-7	2.61e-8	3.31e-8
40 000	4.69e+0	6.25e-3	9.19e+2	9.35e-9	8.74e-8	4.71e-8	6.47e-8
90 000	1.28e+1	1.27e-2	1.51e+3	—	—	7.98e-8	1.25e-7
160 000	2.87e+1	1.38e-2	2.15e+3	—	—	9.02e-8	1.84e-7
250 000	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 $\tilde{A}_n^{-1}$

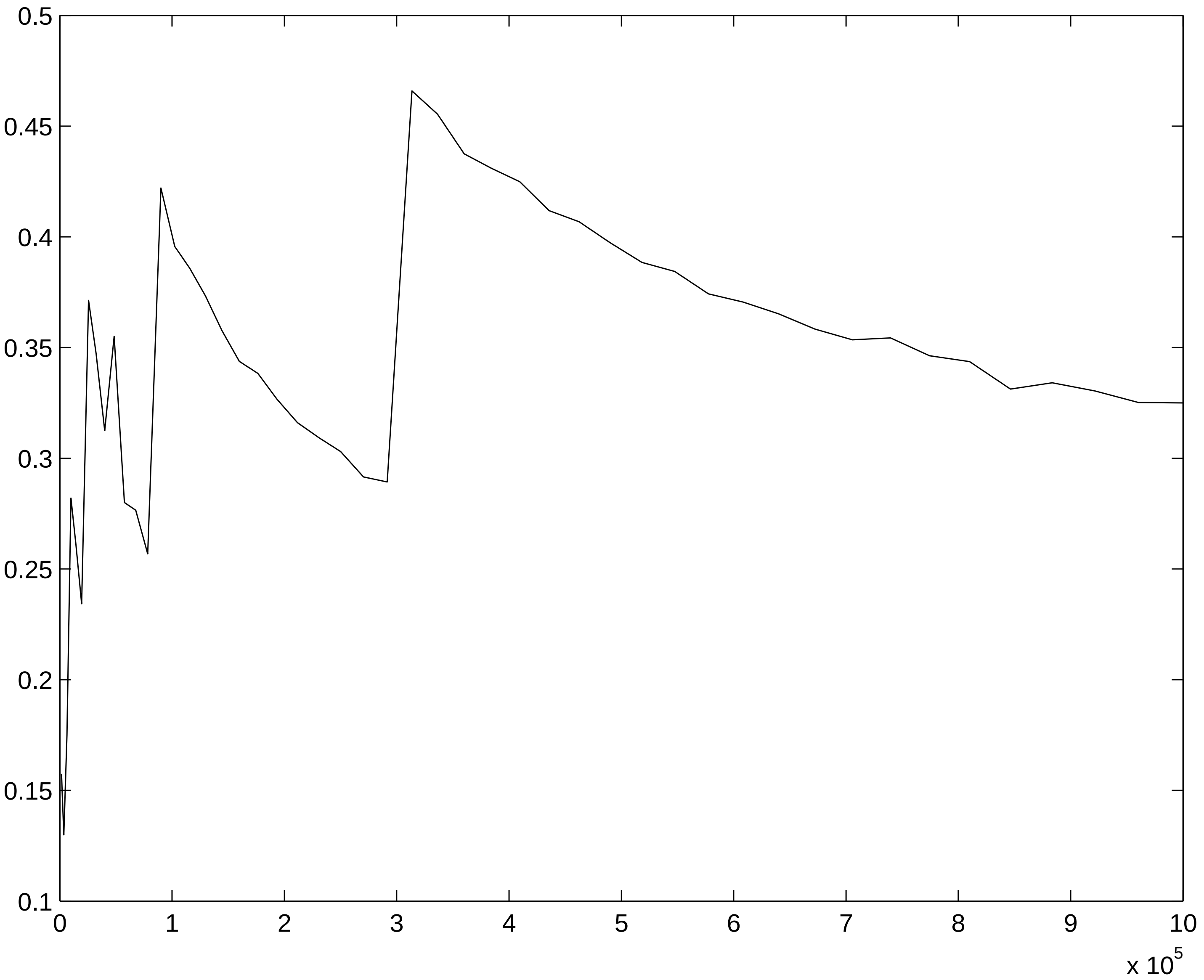
e_2 The error in l^2-operator norm of $\tilde{A}_n^{-1}$

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

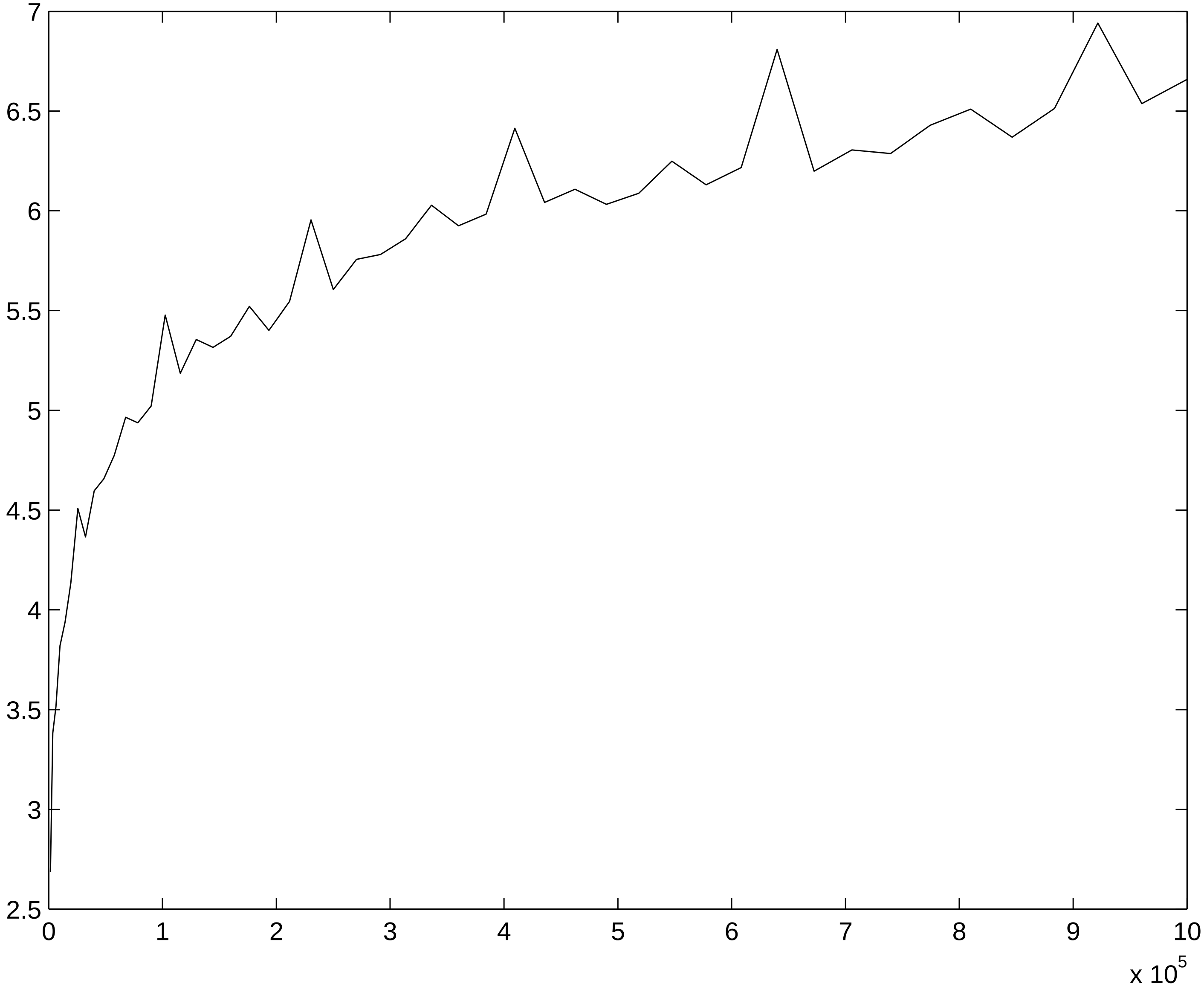
e_4 The l^2-error in the first column of $\tilde{A}_{nn}^{-1}$.



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



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



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

Recall that the inversion scheme relies crucially on the fact that off-diagonal blocks of the system matrix can be approximated by matrices of low rank.

We will next describe how randomized sampling can be used to efficiently construct such low rank approximations.

Algorithm 1: *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 x_1, x_2, $\dots$ be a sequence of vectors in $\mathbb{R}^n$ whose entries are i.i.d. random variables drawn from a normalized Gaussian distribution.

Form the length-m vectors

$$y_1 = A\,x_1, \qquad y_2 = A\,x_2, \qquad y_3 = A\,x_3, \qquad \dots$$

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 1:

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 $x_1,\, x_2,\, \dots$ 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

$$y_1 = A\,x_1, \qquad y_2 = A\,x_2, \qquad y_3 = A\,x_3, \qquad \dots$$

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.

Let us quantify how well the randomly generated vectors span the column space of A:

Recall:

- A is an $m \times n$ matrix.
- $x_1,\, x_2,\, \dots$ are Gaussian random vectors in $\mathbb{R}^n$.
- $y_1 = A\,x_1,\, y_2 = A\,x_2,\, \dots$

For a given integer l, we orthogonalize the vectors $[y_1,\, y_2,\, \dots\,,\, y_l]$,

$$Q_l\, R_l\, P_l = [y_1,\, y_2,\, \dots\,,\, y_l].$$

A measure for how well the vectors in Q_l span A is, for instance,

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

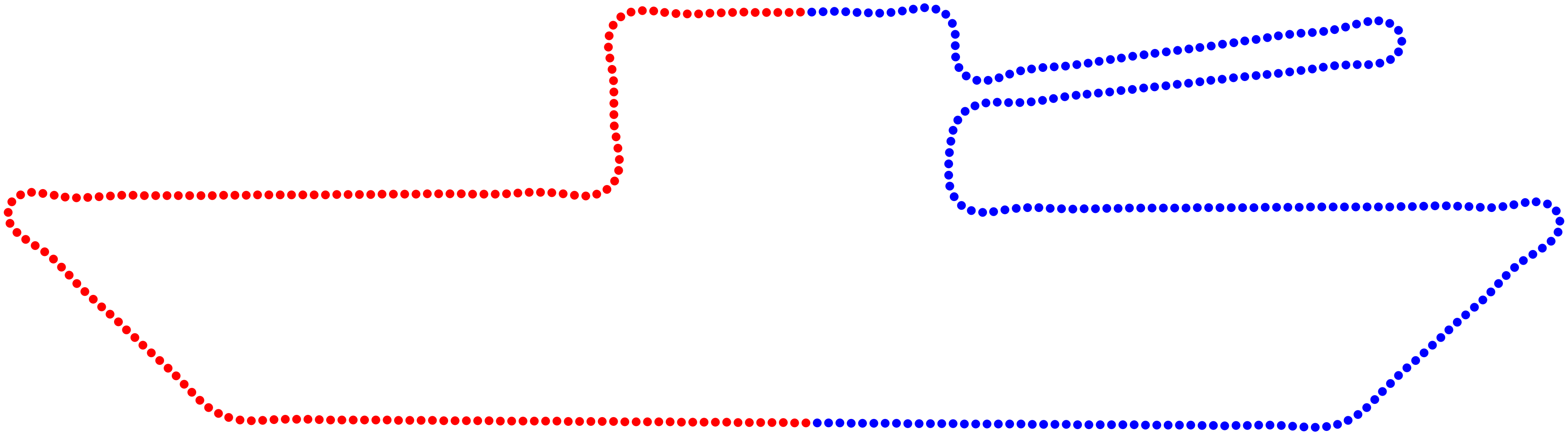
The singular values $\{\sigma_j\}_{j=1}^{n}$ of A provide lower bounds: $e_l \geq \sigma_{l+1}$.

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

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

We will illustrate how e_l and f_l compare to σ_{l+1} with numerical examples.

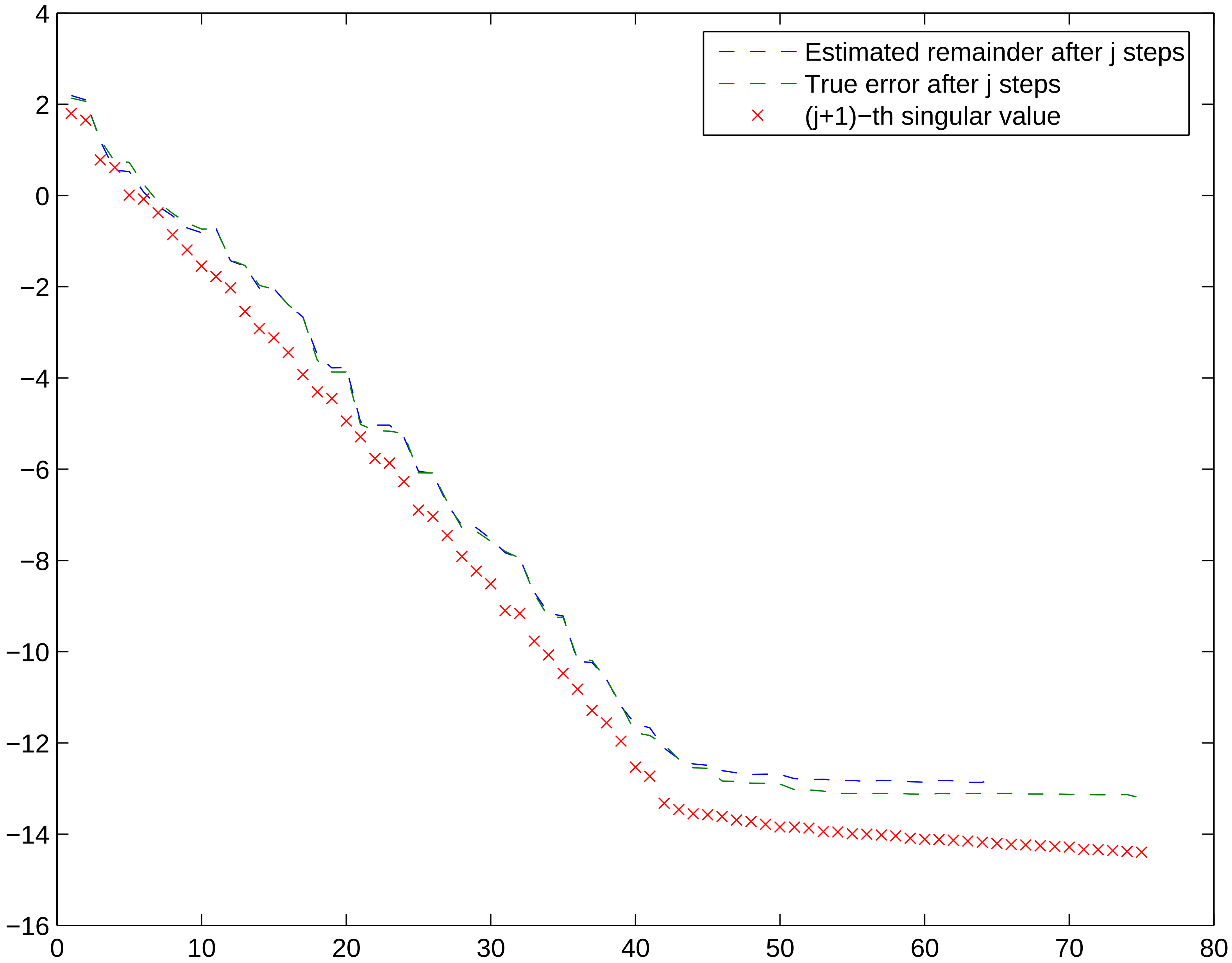
Example: Let A be an off-diagonal block in the double layer potential for the following contour:



We partition the system matrix K into four blocks:

$$K = \begin{bmatrix} \color{red}{X} & A \\ B & \color{blue}{Y} \end{bmatrix}.$$

We seek a low-rank approximation of A.



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

Was this just a lucky realization?

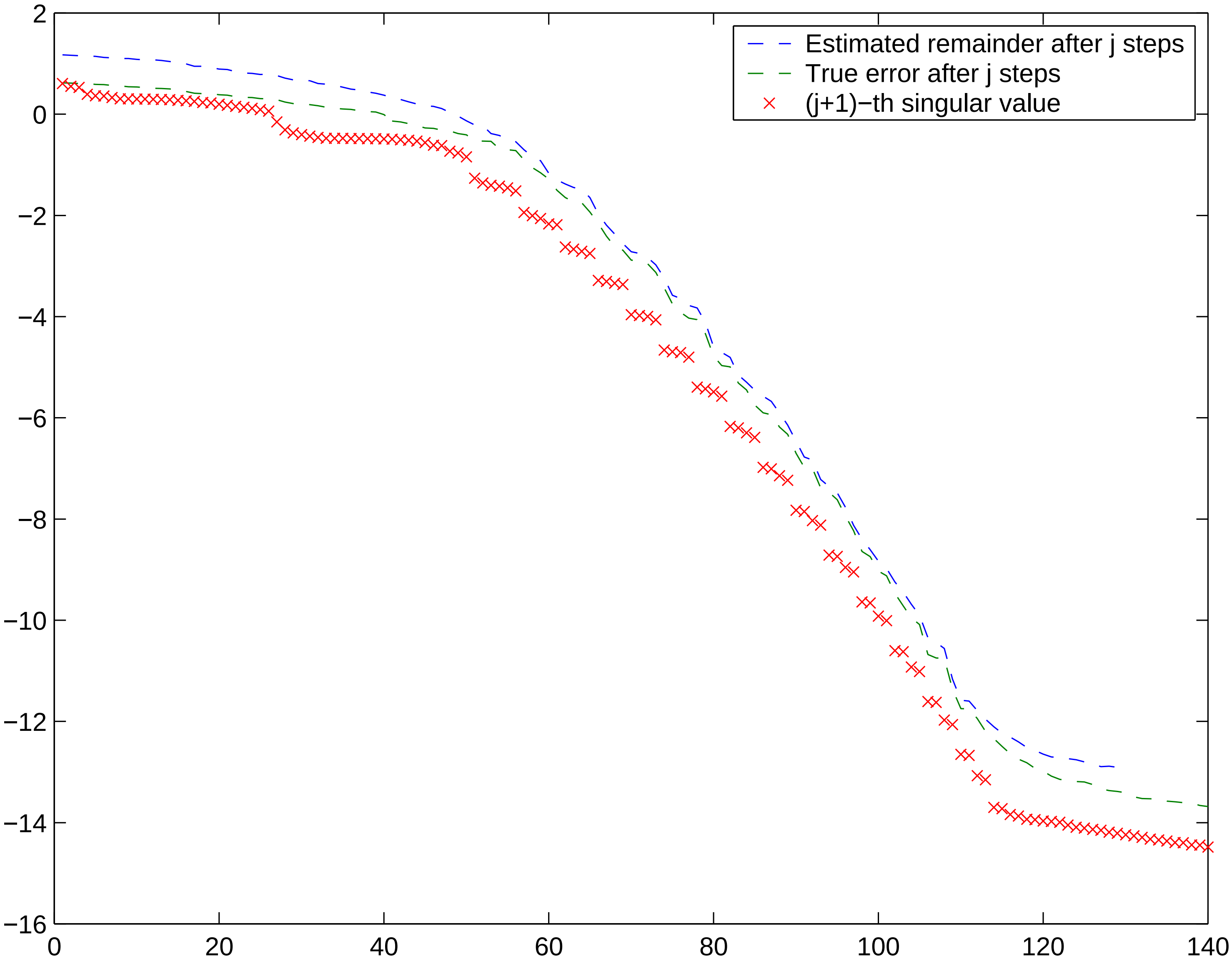
We collected statistics from 1 000 000 realizations:

(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 $= 106$.

These methods do not have "Monte Carlo"-like performance.

The only similarity is in the use of randomness to address non-random problems.

When the randomized sampling method works, it is very highly accurate.
Relative accuracy of 10^{-10} is typical.

The likelihood of failure can inexpensively be rendered entirely negligible.
It is a user defined parameter that can be set to something like 10^{-5} or 10^{-17}.
Cheap verification schemes can reduce it even further (for the truly paranoid).

Note:

Once you have a basis for the column space, you can cheaply get any factorization you want.

To see this, suppose that Q is an orthonormal basis for the column space of A;

$$A = Q\,Q^{\mathrm{t}}\,A.$$

Then compute $Q^{\mathrm{t}}\,A$ and then compute the SVD of this $k \times n$ matrix:

$$Q^{\mathrm{t}}\,A = \tilde{U}\,D\,V^{\mathrm{t}}.$$

Then

$$A = Q\,\left(Q^{\mathrm{t}}\,A\right) = \underbrace{Q\,\tilde{U}}_{=:U}\,D\,V^{\mathrm{t}} = U\,D\,V^{\mathrm{t}}.$$

In many environments, it is not even necessary to compute $Q^{\mathrm{t}}\,A$ ...

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

Let Q be an $m \times l$ matrix whose columns form an ON-basis for the columns of AG.

Let σ_{k+1} denote the $(k+1)$'th singular value of A.

Then

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

with probability at least

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

where φ is a decreasing function satisfying

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

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

Recall the error bound:

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

The high-lighted factor is somewhat undesirable for a couple of reasons:

- The algorithm cannot determine the ε-rank if ε is too close to the computational precision.

- There could be problems in cases where the singular values decay slowly.

Important: In the applications that we have in mind, the singular values decay exponentially. In such cases, the only effect of the $\sqrt{lm}$ factor is that a couple too many random vectors may be generated. *The computed decomposition is still accurate to precision ε.*

How does Algorithm I perform when we do not have a fast method for applying A to a vector?

When $k \ll \min(m, n)$, Algorithm 1 might be slightly faster than Gram-Schmidt:

$$\begin{array}{lll} \text{Multiplications required for Algorithm 1:} & m\,n\,(k+10) & +O(k^2(m+n)). \\ \text{Multiplications required for Gram-Schmidt:} & m\,n\,2\,k & +O(k^2(m+n)). \end{array}$$

Other potential benefits:

- Data-movement.
- Parallelization.

However, many environments remain in which there is little or no gain.

Algorithm 2: An $O(m\,n\,log(k))$ algorithm for *general* matrices:

Work by Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert.
(The speaker was — much to his regret — not involved with this development.)

Recall that Algorithm 1 determines a basis for the column space from the matrix

$$\begin{array}{cccc} Y & = & A & G. \\ m\times l & & m\times n & n\times l \end{array}$$

Key points:

- The product $x \mapsto A\,x$ can be evaluated rapidly.
- The entries of G are i.i.d. random numbers.

What if we do *not* have a fast algorithm for computing $x \mapsto A\,x$?

New idea: Construct G with "some randomness" and "some structure".
Then for each $1 \times n$ row a of A, the matrix-vector product

$$a \mapsto a\,G$$

can be evaluated using $n\,\log(l)$ operations.

What is this “random but structured” matrix G?

$$\begin{array}{ccccc} G & = & D & F & S \\ n \times l & & n \times n & n \times n & n \times l \end{array}$$

where,

- D is a diagonal matrix whose entries are i.i.d. random variables drawn from a uniform distribution on the unit circle in $\mathbb{C}$.

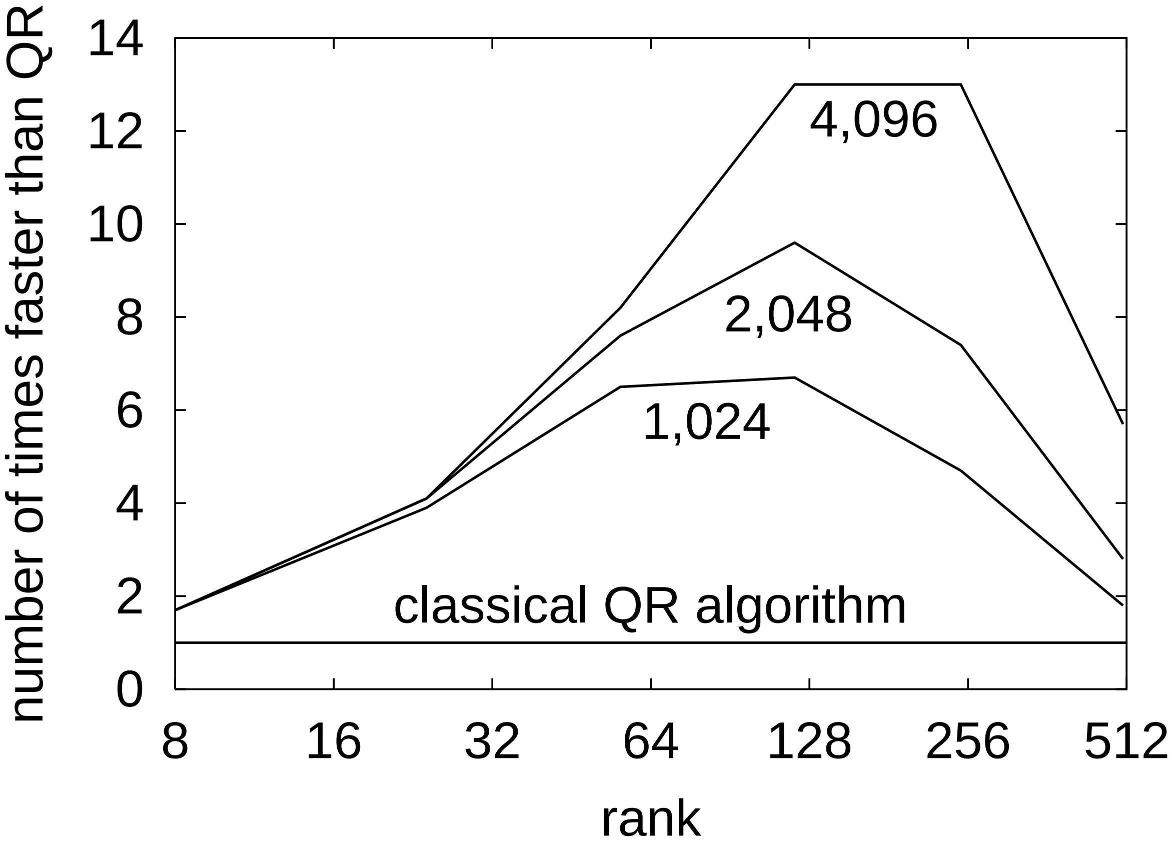
- F is the discrete Fourier transform, $F_{jk} = e^{-2\pi i(j-1)(k-1)/n}$.

- S is a matrix whose entries are all zeros except for a single, randomly placed 1 in each column. (In other words, the action of S is to draw l columns at random from $D\,F$.)

Note: Other successful choices of the matrix G have been tested, for instance, the Fourier transform may be replaced by the Walsh-Hadamard transform.

This idea was described by Nir Ailon and Bernard Chazelle (2006).
There is also related recent work by Sarlós (on randomized regression).

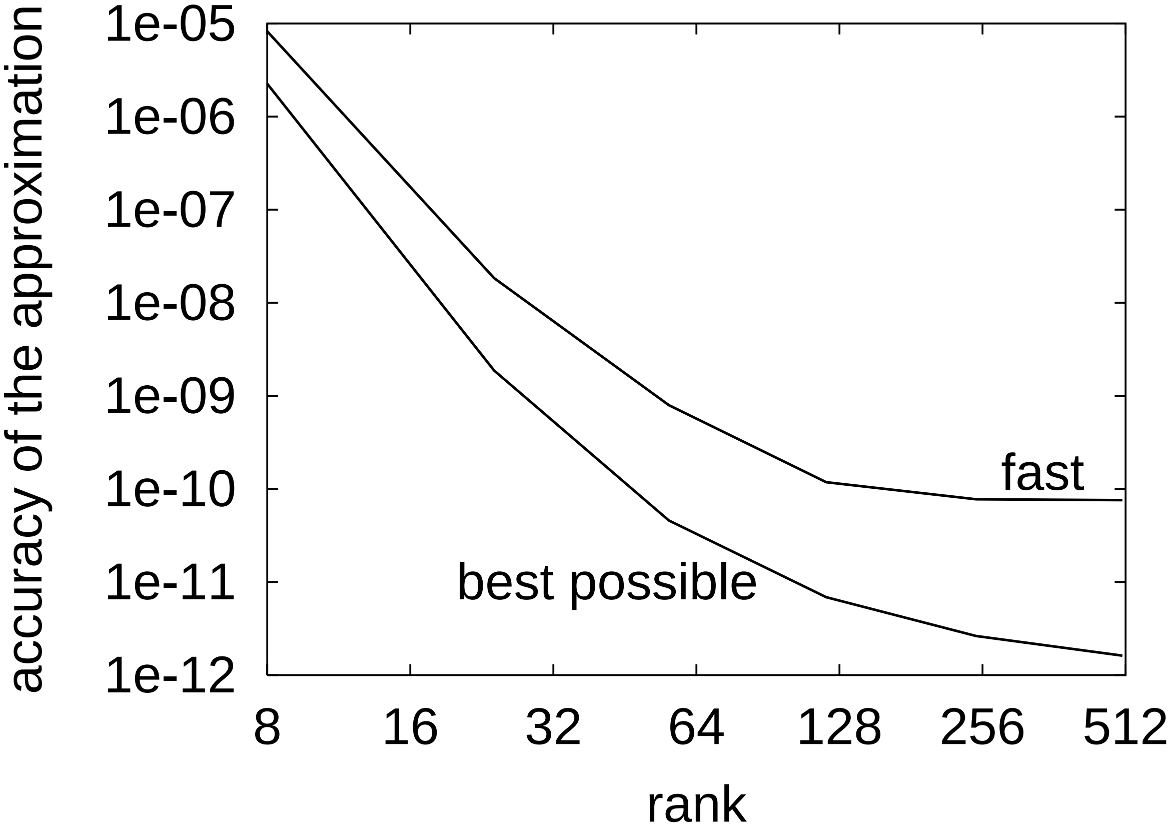
Speed gain for square matrices of various sizes



The time required to verify the approximation is included in the fast, but not in the classical timings.

This slide comes from a talk by Mark Tygert.

Empirical accuracy on 2,048-long convolution



The estimates of the accuracy of the approximation are accurate to at least two digits of relative precision.

This slide comes from a talk by Mark Tygert.

What theory do we have for Algorithm 2? (The $O(m\,n\,\log(k))$ one.)

It is less well developed than for Algorithm 1.

The core question is to quantify the "amount of randomness" in the matrix G. We do not need all entries to be i.i.d. Gaussian random numbers, but what exactly do we need?

Experiments show that for some choices of G, the computational results are almost indistinguishable from the fully random case. Currently, we do not have theory to back this observation up.

It is time to definitively deal with linear boundary value problems:

- We need to develop machinery for dealing with surfaces.
- We need faster and more robust solvers.

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. Theory mostly finished. (Or is it?)

Randomized sampling:

- More stable in the Lanczos environment. Probably faster too.
- For general matrices, it is $O(m\, n \log(k))$.
- Very interesting tool for coarse graining in physical sciences.
- Applications to network analysis, data mining, fast solvers *etc.*