Fast numerical methods for solving elliptic PDEs

Gunnar Martinsson
The University of Colorado at Boulder

Students:

Tracy Babb

JaeAnn Dwulet

Adrianna Gillman

Nathan Halko

Patrick Young

Collaborators:

Vladimir Rokhlin (Yale)

Joel Tropp (Caltech)

Mark Tygert (Courant)

- *Methods for discretizing linear elliptic partial differential equations.*
  How do you approximate a linear boundary value problem such as, *e.g.*,

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

  by a linear $N \times N$ system $\mathsf{A}\,\boldsymbol{u} = \boldsymbol{b}$ suitable for solving on a computer?

- *Fast solvers for linear systems associated with linear BVPs.*
  New: In many cases, A^{-1} can be computed in $O(N)$ operations.

- *Methods for computing approximate factorizations of low-rank matrices.*
  Given an $m \times n$ matrix X of ε-rank k, how do you compute a factorization

$$\underset{m \times n}{\mathsf{X}} \quad \approx \quad \underset{m \times k}{\mathsf{Y}} \quad \underset{k \times n}{\mathsf{Z}}.$$

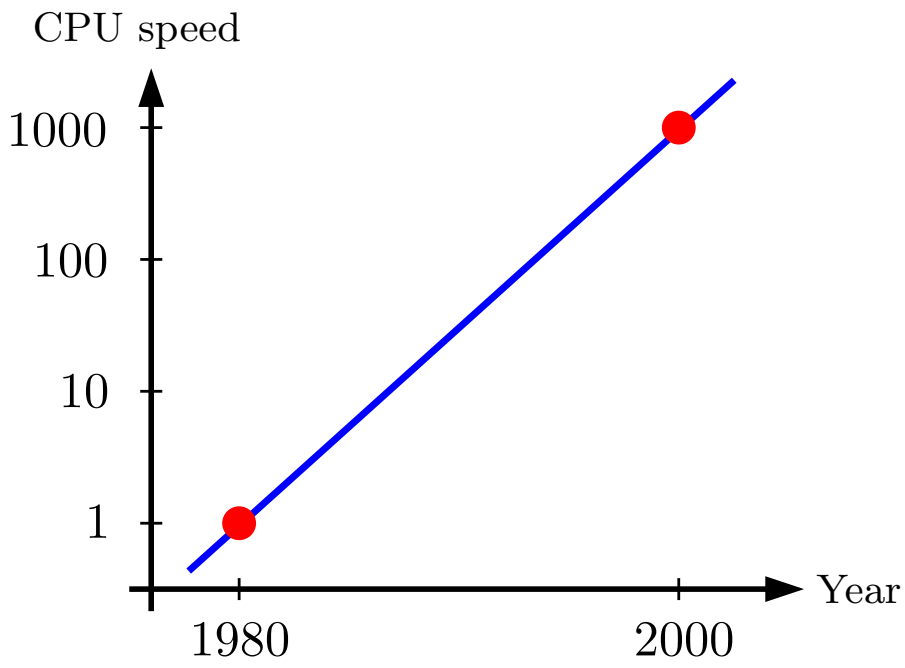
  New: Randomized methods that do this in $O(m\,n\,\log(k))$ operations,
  or does it in $O(m\,n\,k)$ operations, but with a single pass over the data.

Definition of the term "fast":

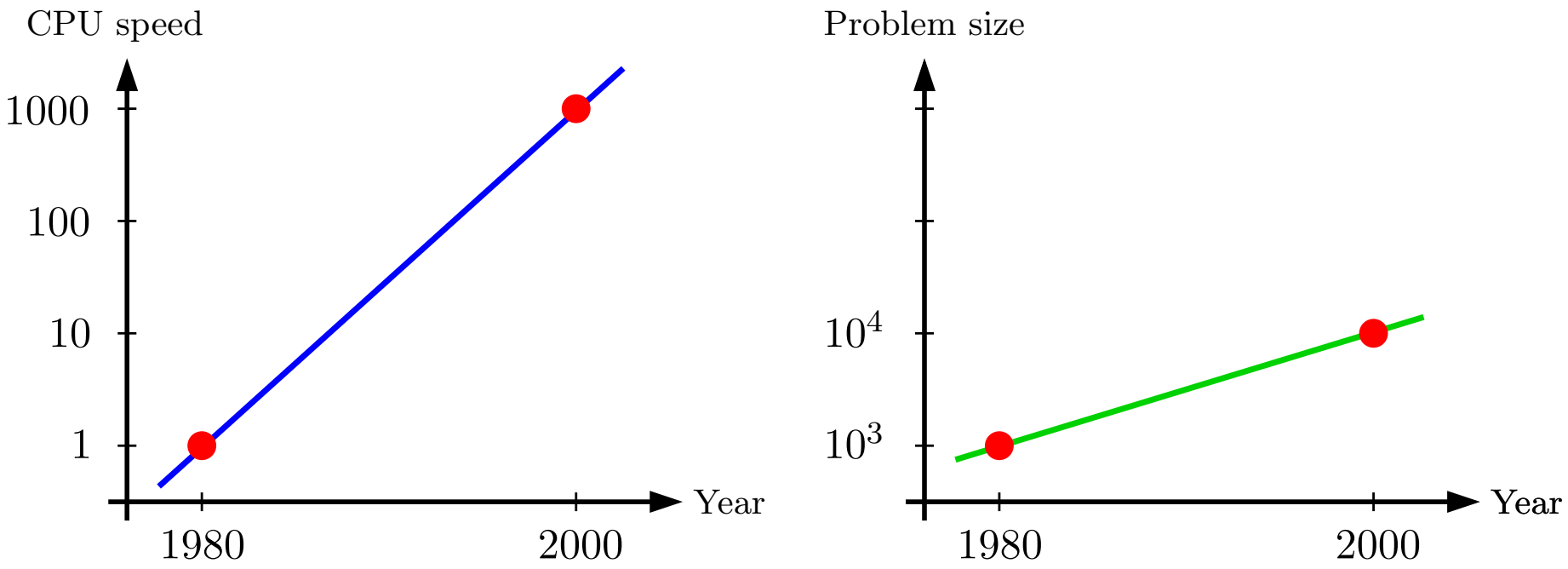
We say that a numerical method is *fast* if its execution time scales as $O(N)$ as the problem size N grows.

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

Growth of computing power and the importance of algorithms



Growth of computing power and the importance of algorithms

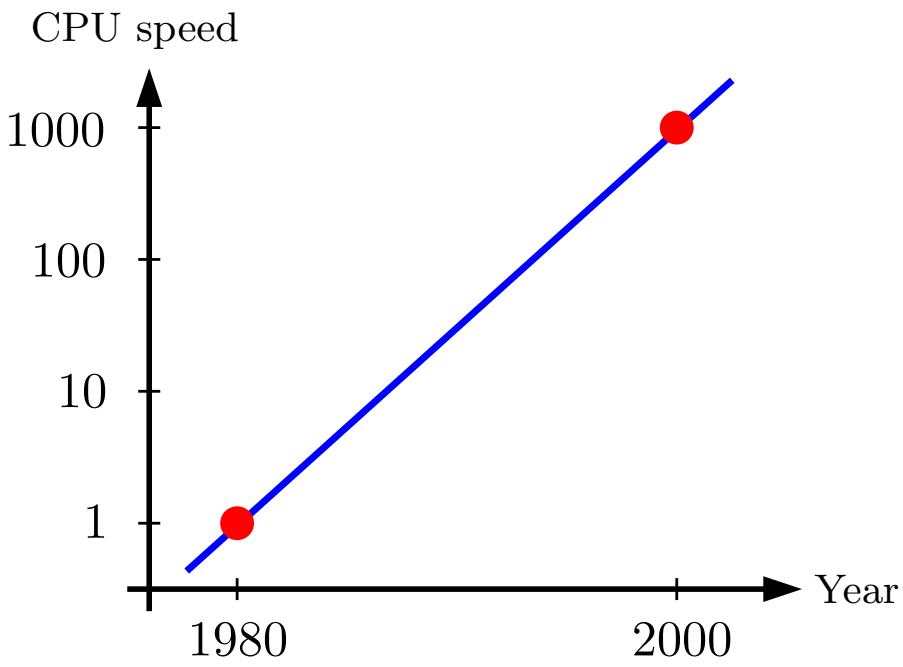


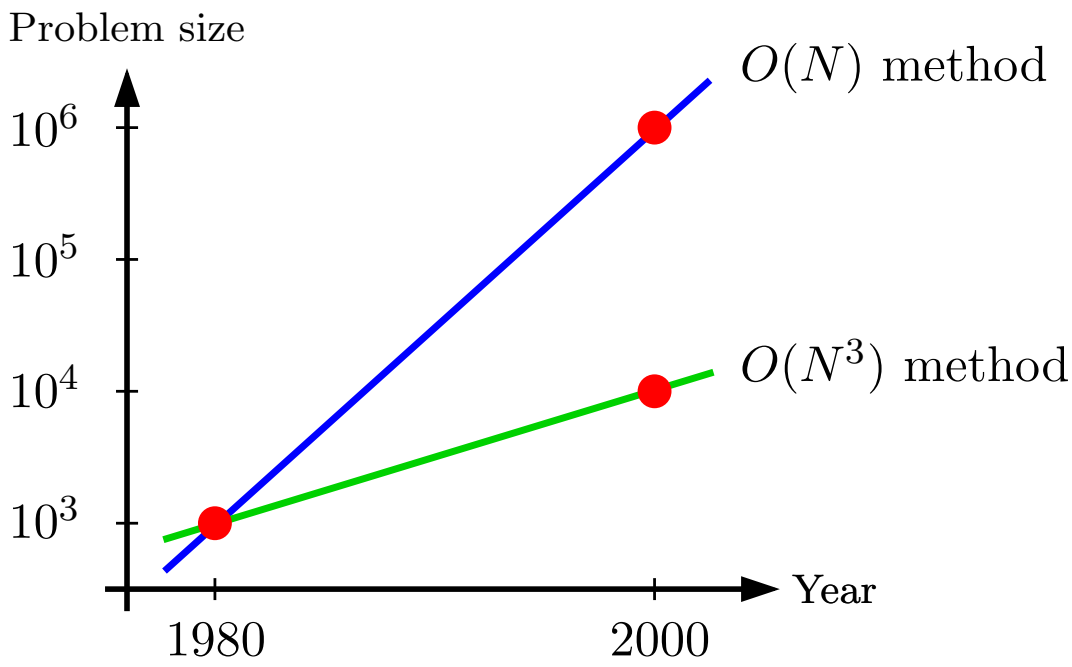
Consider the computational task of solving a linear system $\mathsf{A}\,\boldsymbol{u} = \boldsymbol{b}$ of N algebraic equations with N unknowns.

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 $\mathsf{A}\,\boldsymbol{u} = \boldsymbol{b}$ 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.

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.*

In consequence, successful algorithms must scale well both with problem size and with the number of processors that a computer has.

To slightly offset the difficulty of parallelization, the *cost of storage is decreasing.* However, the speed of access is increasing only slowly, again reinforcing the need to keep data local in designing algorithms.

We will demonstrate that *randomized algorithms* form an excellent tool in minimizing communication constraints.

Discretization of linear Boundary Value Problems

We consider stationary linear Boundary Value Problems of the form

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

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

- 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: Laplace's equation with Dirichlet boundary data:

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

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 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, $O(N)$.

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 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.

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

$$\mathsf{A}\,\boldsymbol{u} = \boldsymbol{b}, \tag{2}$$

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

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

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

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

The matrix T is typically constructed in a compressed format that allows the matrix-vector product $\mathsf{T}\,\boldsymbol{b}$ to be evaluated rapidly.

Variation: Find factors B and C such that $||\mathsf{A} - \mathsf{B}\,\mathsf{C}|| \leq \varepsilon$, and linear solves involving the matrices B and C are fast.

"Iterative" versus "direct" solvers

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

$$\mathsf{A}\,\boldsymbol{u} = \boldsymbol{b}.$$

Iterative methods:

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

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

Many iterative methods access 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 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 possible.

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.

   Scattering problems intractable to existing methods can 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 should be viewed as a step towards putting together a LAPACK-type environment for solving the basic linear boundary value problems of mathematical physics.

Origins of fast direct solvers:

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, 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.*

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

Current status — problems with non-oscillatory kernels (Laplace, elasticity, *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:* Some $O(N\,(\log N)^p)$ inversion algorithms exist. Work remains — general grids, improve constants, *etc*.
- *Volume Int. Eq. in the plane (e.g. low frequency Lippman-Schwinger):* $O(N\,(\log N)^p)$ inversion algorithms exist. Implementation is under way.
- *Boundary Integral Equations in* $\mathbb{R}^3$*:* $O(N^{3/2})$ techniques exist. $O(N \log N)$ techniques are being developed.

Problems on 3D domains:

- ????

  (Memory requirements is a major concern.)

Current status — problems with oscillatory kernels (Helmholtz, Maxwell, *etc*).

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:* ???
  ($O(N^{3/2})$ inversion is possible.)
- *Volume Int. Eq. in the plane (e.g. high frequency Lippman-Schwinger):* ???
  ($O(N^{3/2})$ inversion is possible.)
- *Boundary Integral Equations in* $\mathbb{R}^3$*:* ???

Problems on 3D domains:

- ????
  ($O(N^2)$ inversion possible — but again, memory requirements is a concern.)

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.

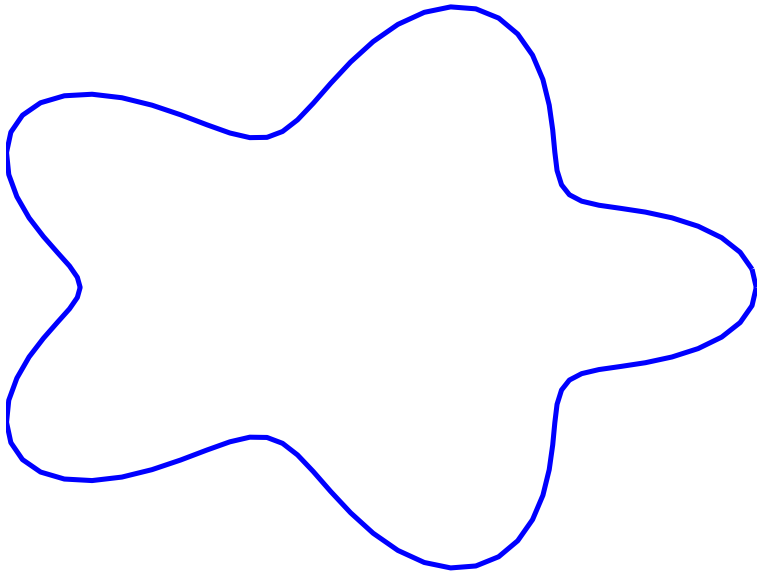
We start by describing some key properties of the matrices under consideration.

For concreteness, consider a 100×100 matrix A approximating the operator

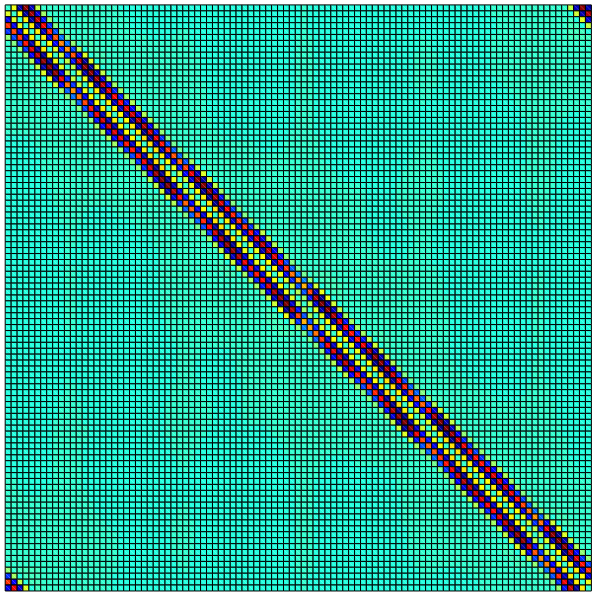
$$[\mathcal{S}_\Gamma\, u](\boldsymbol{x}) = u(\boldsymbol{x}) + \int_\Gamma \log|\boldsymbol{x} - \boldsymbol{y}|\, u(\boldsymbol{y})\, ds(\boldsymbol{y}).$$

The matrix A is characterized by:

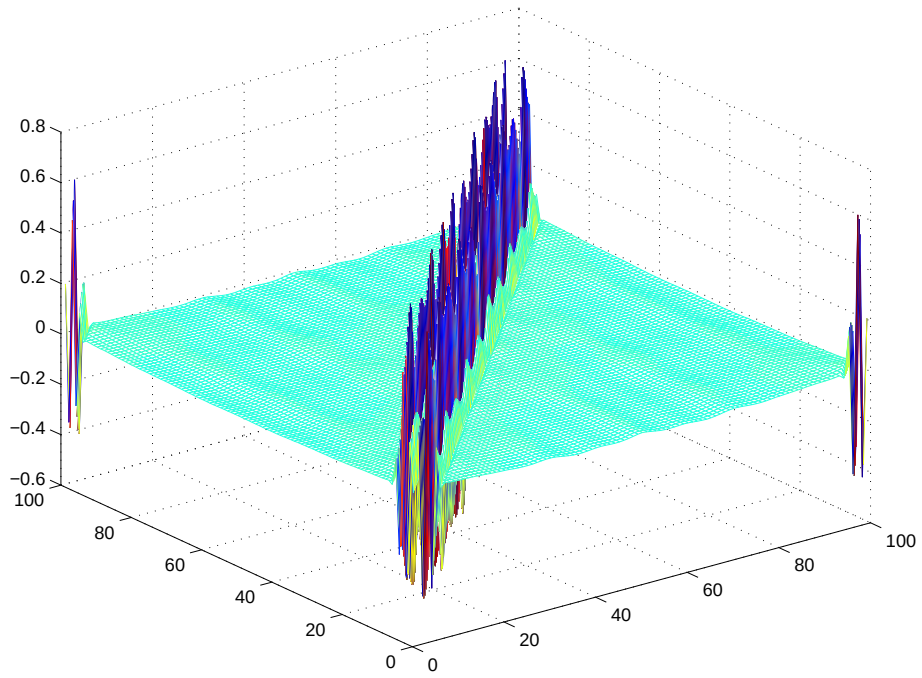
- Irregular behavior near the diagonal.
- Smooth entries away from the diagonal.



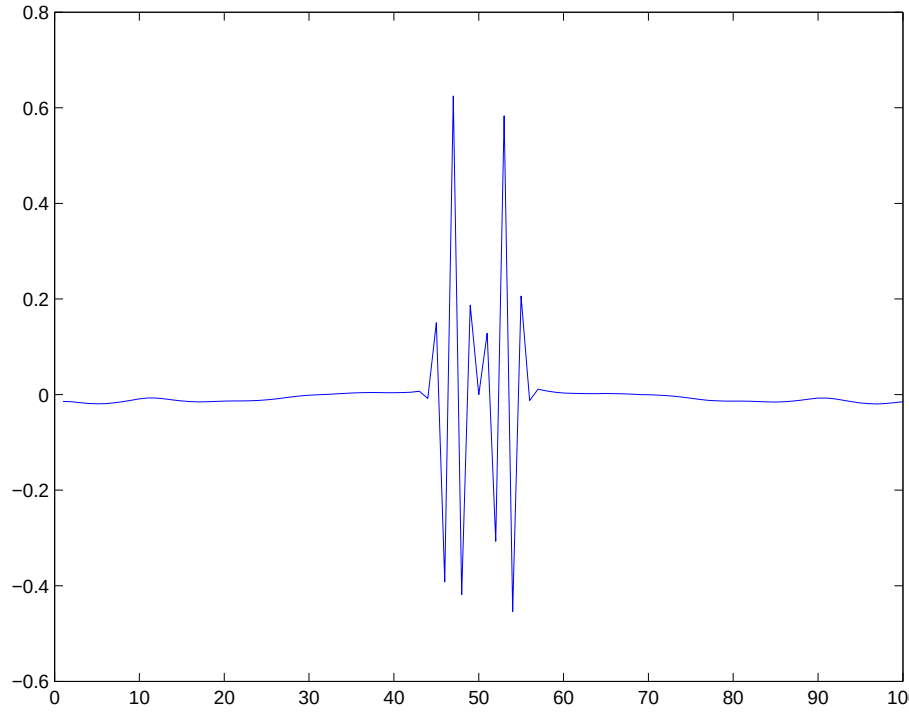
The contour Γ.



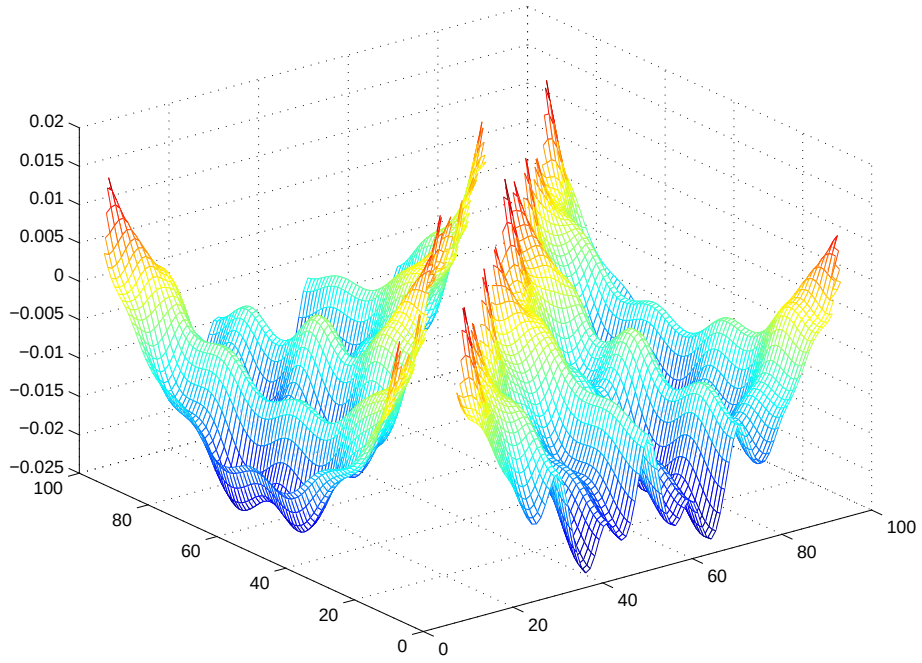
The matrix A.



Plot of a_{ij} vs i and j

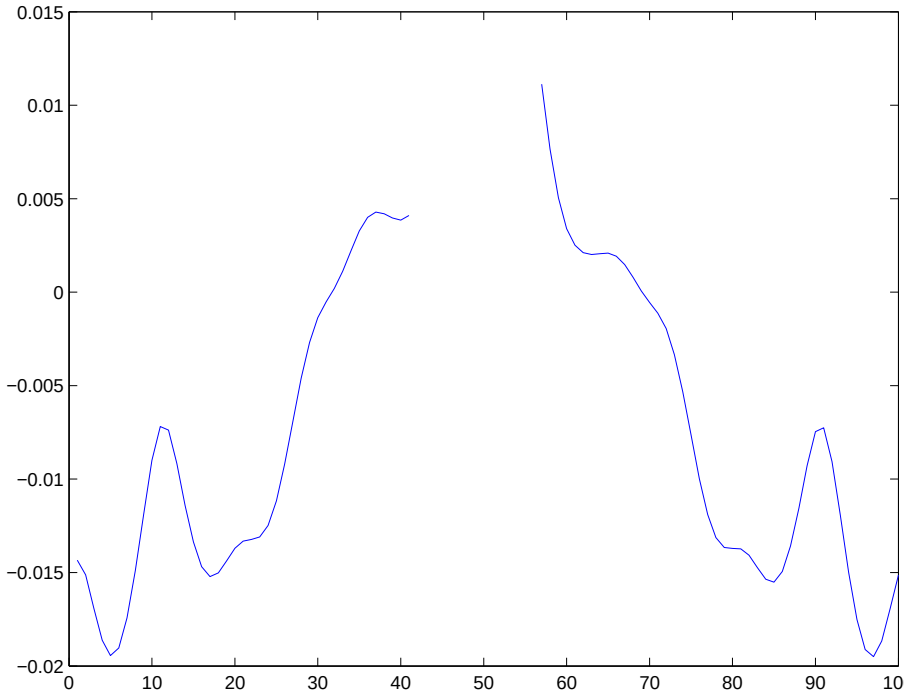


The 50^{th} row of A



Plot of a_{ij} vs i and j

(without the diagonal entries)

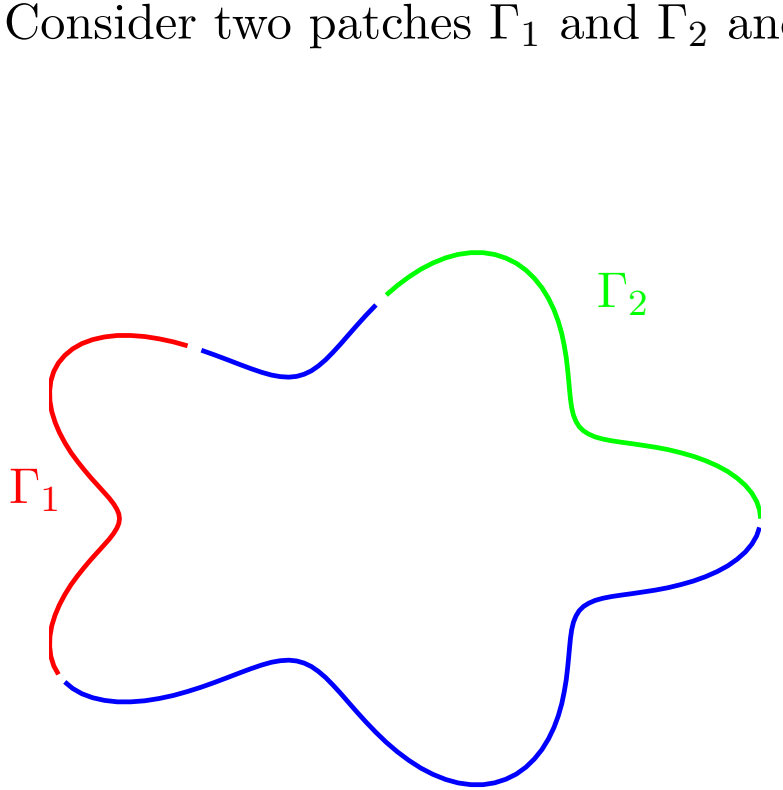


The 50^{th} row of A

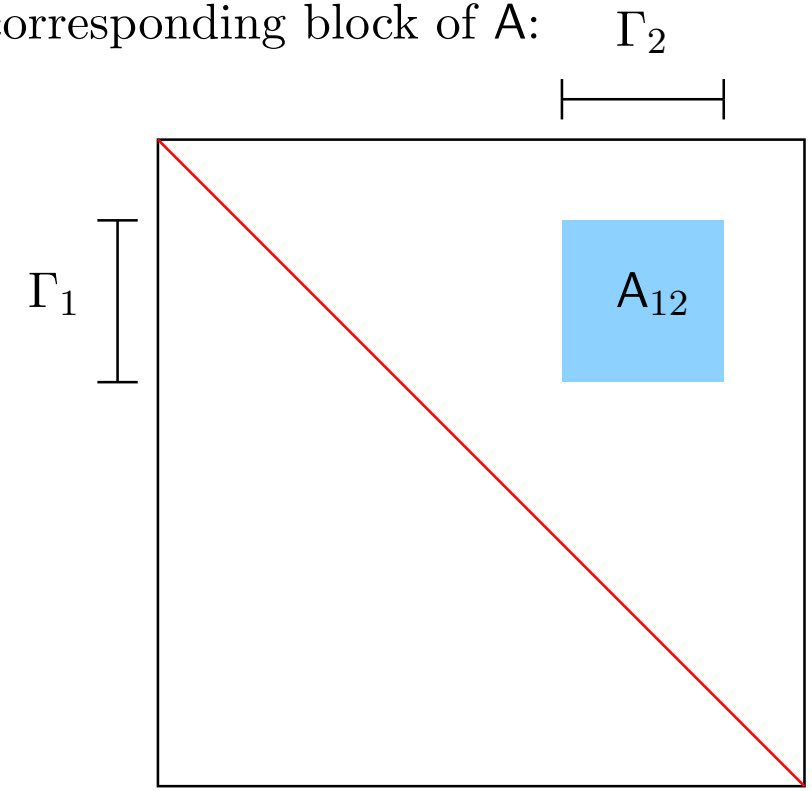
(without the diagonal entries)

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

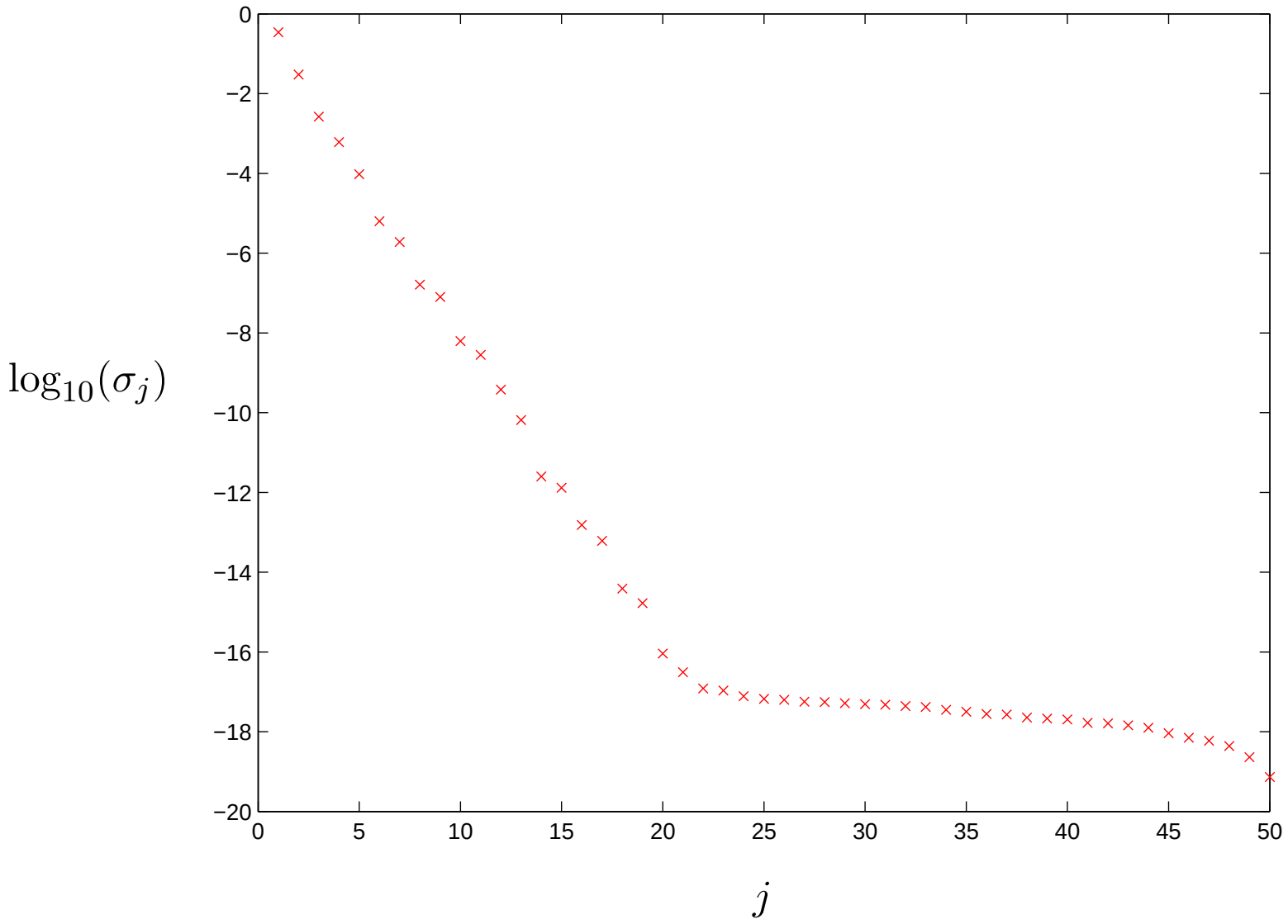


The matrix A

The block A_{12} is a discretization of the integral operator

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

Singular values of A_{12} (now for a 200×200 matrix 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.*

Let A be a matrix consisting of $p \times p$ blocks of size $n \times n$:

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

Core assumption: Each off-diagonal block A_{ij} admits the factorization

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

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 U_i for their column spaces (and analogously all blocks in the j'th column use the same basis matrices V_j for their row spaces).

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

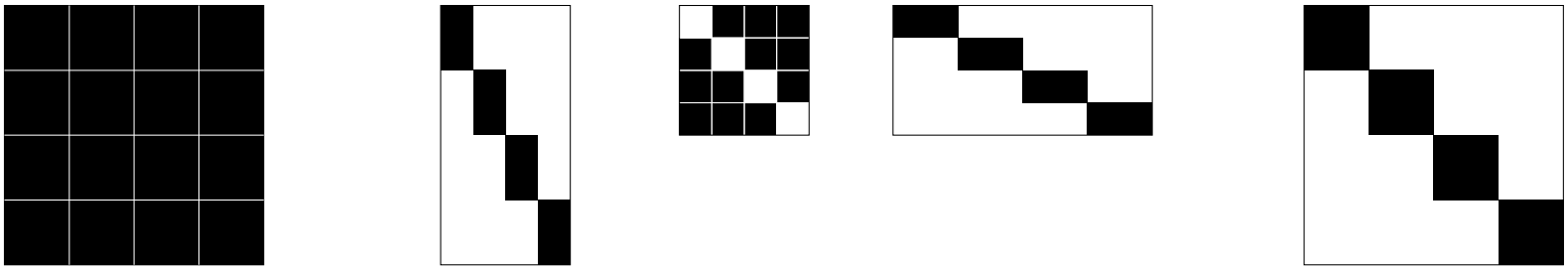
Then A admits the factorization:

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

or

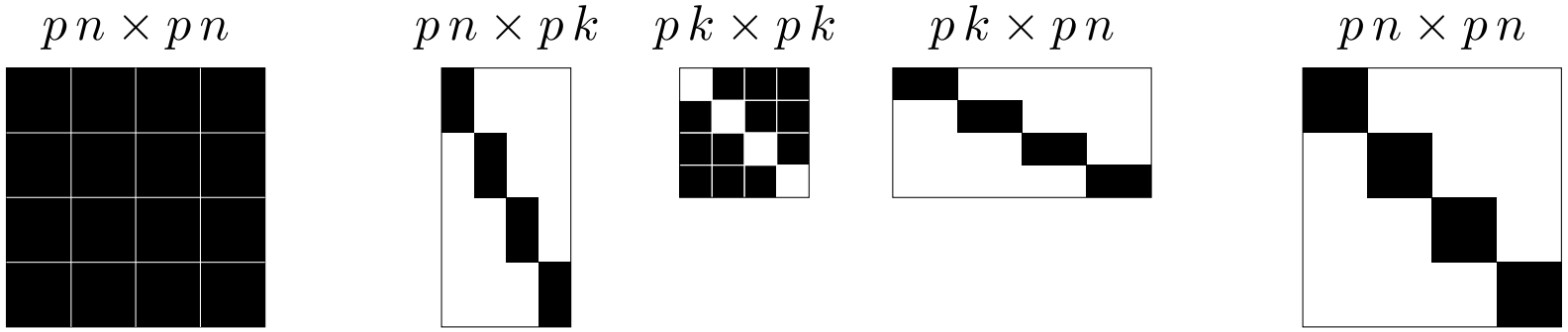
$$\mathsf{A} = \mathsf{U}\ \tilde{\mathsf{A}}\ \mathsf{V}^* + \mathsf{D},$$

$p\,n \times p\,n$ $\quad$ $p\,n \times p\,k$ $\quad$ $p\,k \times p\,k$ $\quad$ $p\,k \times p\,n$ $\quad$ $p\,n \times p\,n$



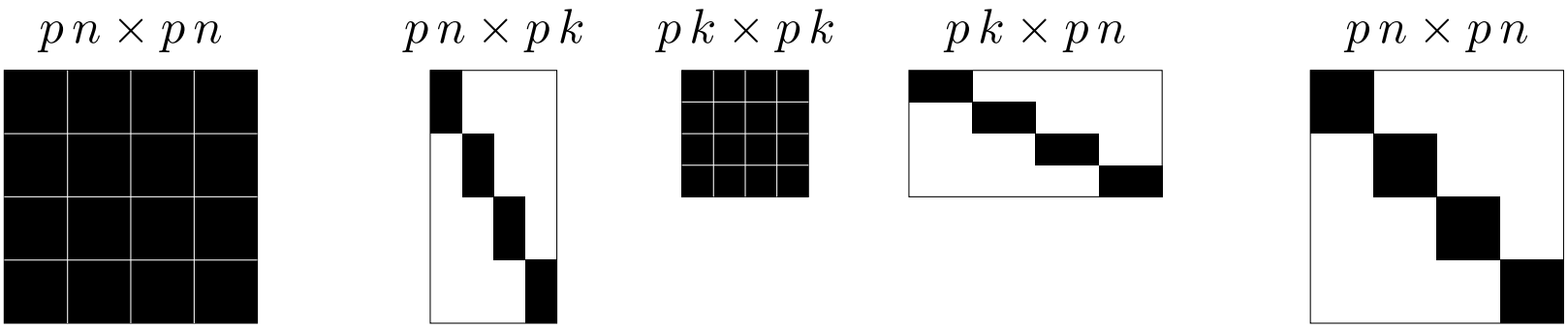
Lemma: [Variation of Woodbury] If an $N \times N$ matrix A admits the factorization

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



then

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



where (provided all intermediate matrices are invertible)

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

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: A has $p \times p$ blocks, each of size $n \times n$ and of rank k.)

We must recurse!

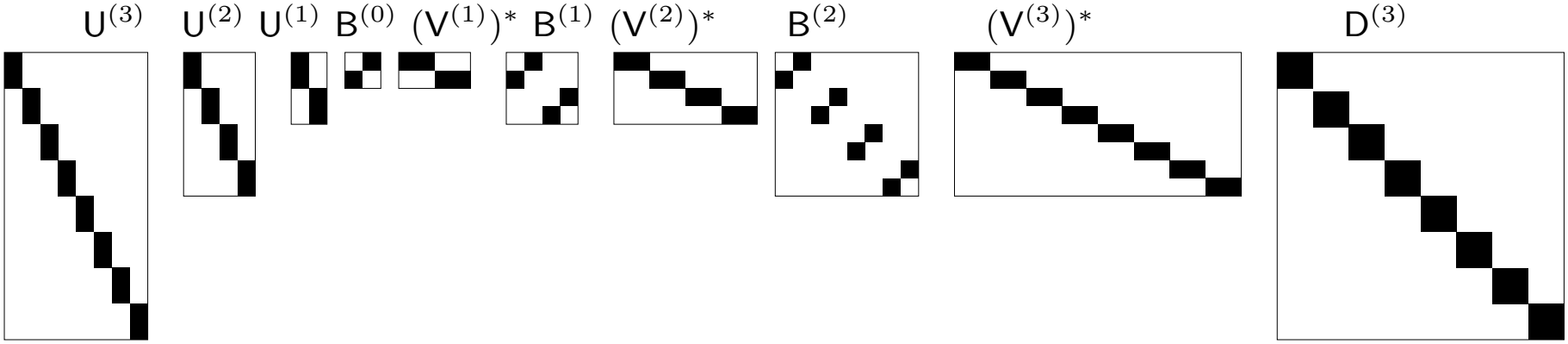
Using a telescoping factorization of A:

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

we have a formula

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

Block structure of factorization:



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

Many details are left out ...

- How do you represent potentials?
  Multipole expansions, proxy charges, interpolation, ...
  Choosing the right one can reduce the run-time by a factor of 10 or more.

- Generalization to
  - Volume integral equations in $\mathbb{R}^2$.
  - Boundary integral equations in $\mathbb{R}^3$.
  - Volume integral equations in $\mathbb{R}^3$. (Currently practical only in certain environments.)

- How do you compute the low-rank approximations in the telescoping factorization in the first place?
  - Note that in "traditional" fast methods such as the *Fast Multipole Method*, the kernel is known and given in analytic form.
  - In a direct method, determining the kernel is part of the problem.

Randomized methods for computing kernel approximations (and lots of other things too)

We cast the kernel approximation problem as a *low-rank approximation problem:*

> ***Primitive problem:*** Given an $m \times n$ matrix A and an integer $\ell < \min(m, n)$, find an orthonormal $m \times \ell$ matrix Q such that $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

In other words, the columns of Q form an ON-basis for the range of A.

We want ℓ to be reasonably close to the theoretically minimal number, but it is more important for the approximation to be accurate.

We for now assume that we know the approximate rank in advance; in applications one is typically given a computational tolerance, and determining the rank is part of the problem — we will deal with this complication shortly.

Primitive problem: Given an $m \times n$ matrix A and an integer $\ell < \min(m, n)$, find an orthonormal $m \times \ell$ matrix Q such that $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

Any standard factorization can easily be obtained from Q*.*

To illustrate, suppose that we seek an approximate rank-ℓ SVD

$$\begin{array}{ccccccc} \mathsf{A} & \approx & \mathsf{U} & & \Sigma & & \mathsf{V}^*, \\ m \times n & & m \times \ell & & \ell \times \ell & & \ell \times n \end{array}$$

where U and V are orthonormal, and Σ is diagonal with non-negative entries.

The following steps will do the job:

1. Form the (small) matrix $\mathsf{B} = \mathsf{Q}^*\mathsf{A}$.
2. Compute the SVD of the (small) matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
3. Set $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Note: The *Golub-Businger algorithm* is very similar. In GS, you solve the "primitive problem" via QR. This directly yields a factorization $\mathsf{A} \approx \mathsf{QRP}$; then set $\mathsf{B} = \mathsf{RP}$ in Step 2, and execute Step 3 as described.

Primitive problem: Given an $m \times n$ matrix A and an integer $\ell < \min(m, n)$, find an orthonormal $m \times \ell$ matrix Q such that $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

Solving the primitive problem via randomized sampling — intuition:

1. Draw random vectors $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \cdots \in \mathbb{R}^n$.
   (We will discuss the choice of distribution later — think Gaussian for now.)

2. Form "sample" vectors $\boldsymbol{y}_1 = \mathsf{A}\,\boldsymbol{\omega}_1, \boldsymbol{y}_2 = \mathsf{A}\,\boldsymbol{\omega}_2, \boldsymbol{y}_3 = \mathsf{A}\,\boldsymbol{\omega}_3, \cdots \in \mathbb{R}^m$.

3. Form orthonormal vectors $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3, \cdots \in \mathbb{R}^m$ such that

$$\text{Span}(\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_\ell) = \text{Span}(\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_\ell).$$

   For instance, Gram-Schmidt can be used — pivoting is rarely required.

If A has *exact* rank ℓ, then $\text{Span}\{\boldsymbol{q}_j\}_{j=1}^{\ell} = \text{Ran}(\mathsf{A})$ with probability 1.

Primitive problem: Given an $m \times n$ matrix A and an integer $\ell < \min(m, n)$, find an orthonormal $m \times \ell$ matrix Q such that $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

Solving the primitive problem via randomized sampling — intuition:

1. Draw random vectors $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \cdots \in \mathbb{R}^n$.
   (We will discuss the choice of distribution later — think Gaussian for now.)

2. Form "sample" vectors $\boldsymbol{y}_1 = \mathsf{A}\,\boldsymbol{\omega}_1, \boldsymbol{y}_2 = \mathsf{A}\,\boldsymbol{\omega}_2, \boldsymbol{y}_3 = \mathsf{A}\,\boldsymbol{\omega}_3, \cdots \in \mathbb{R}^m$.

3. Form orthonormal vectors $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3, \cdots \in \mathbb{R}^m$ such that

$$\text{Span}(\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_\ell) = \text{Span}(\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_\ell).$$

   For instance, Gram-Schmidt can be used — pivoting is rarely required.

If A has *exact* rank ℓ, then $\text{Span}\{\boldsymbol{q}_j\}_{j=1}^{\ell} = \text{Ran}(\mathsf{A})$ with probability 1.

What is perhaps surprising is that even in the general case, $\{\boldsymbol{q}_j\}_{j=1}^{\ell}$ often does almost as good of a job as the theoretically optimal vectors (which happen to be the ℓ leading left singular vectors).

Primitive problem: Given an $m \times n$ matrix A and an integer $\ell < \min(m, n)$, find an orthonormal $m \times \ell$ matrix Q such that $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

Randomized algorithm — formal description:

1. Construct a random matrix Ω_ℓ of size $n \times \ell$.
   Suppose for now that Ω_ℓ is Gaussian.

2. Form the $m \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.

3. Construct an $m \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.
   (In other words, the columns of Q_ℓ form an ON basis for $\mathrm{Ran}(\mathsf{Y}_\ell)$.)

Error measure:

The error incurred by the algorithm is $e_\ell = ||\mathsf{A} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}||$.

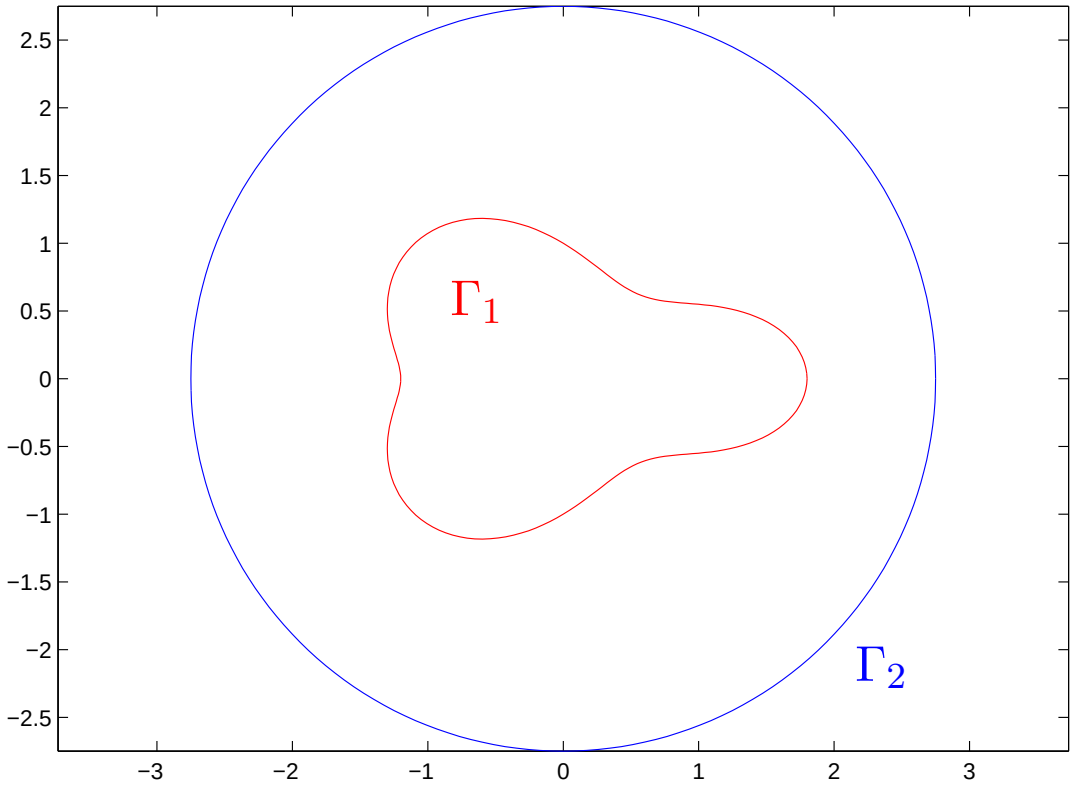
The error e_ℓ is bounded from below by $\sigma_{\ell+1} = \inf\{||\mathsf{A} - \mathsf{B}|| : \mathsf{B} \text{ has rank } \ell\}$.

Specific example to illustrate the performance:

Let A be a 200×200 matrix arising from discretization of

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

where Γ_1 is shown in red and Γ_2 is shown in blue:

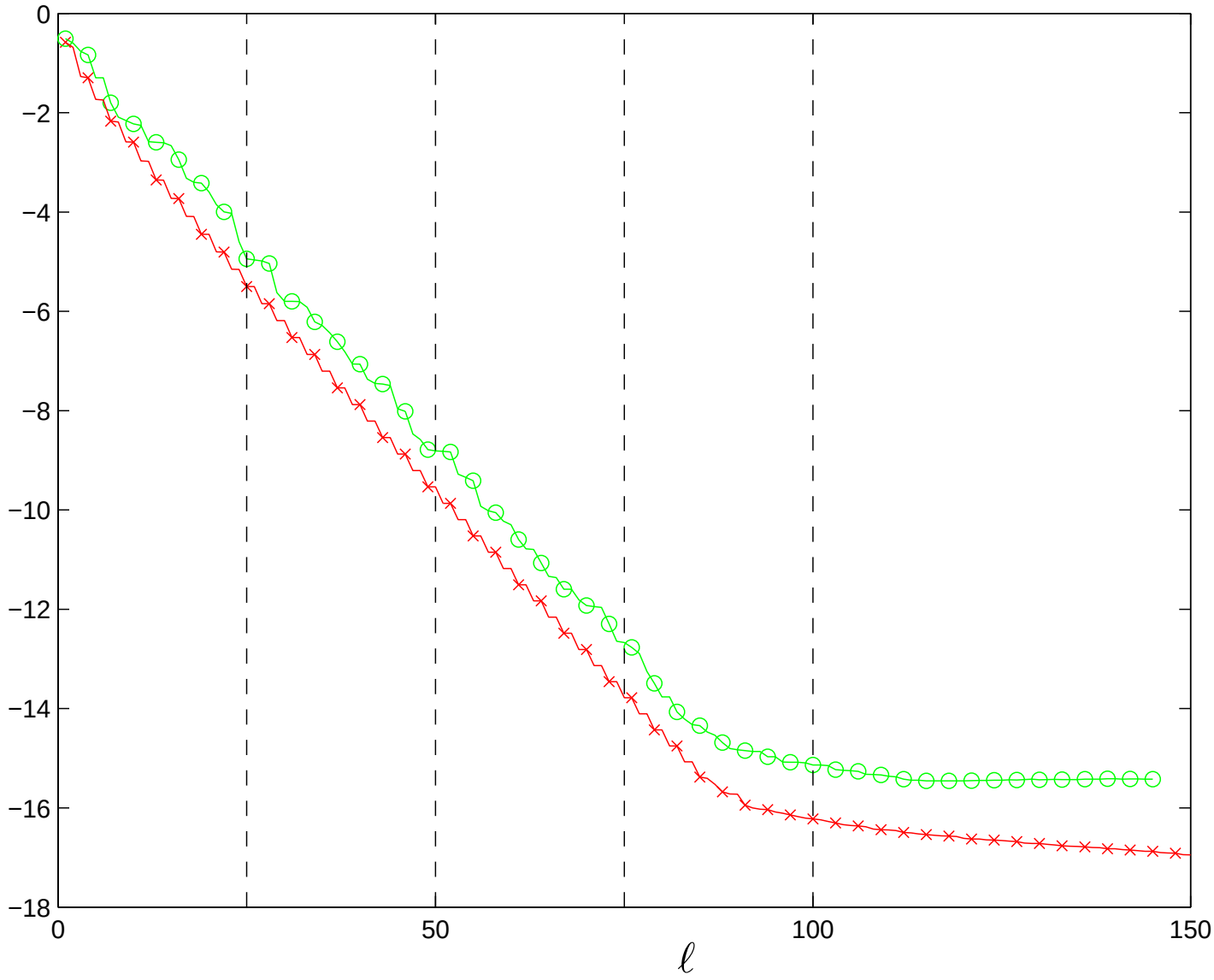


The number α is chosen so that $||\mathsf{A}|| = \sigma_1 = 1$.

Results from one realization of the randomized algorithm

$\log_{10}(e_\ell)$
(actual error)

$\log_{10}(\sigma_{\ell+1})$
(theoretically
minimal error)



Primitive problem: Given an $m \times n$ matrix A and an integer $\ell < \min(m, n)$, find an orthonormal $m \times \ell$ matrix Q such that $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

1. Construct a random matrix Ω_ℓ of size $n \times \ell$.
   Suppose for now that Ω_ℓ is Gaussian.

2. Form the $m \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.

3. Construct an $m \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.
   (In other words, the columns of Q_ℓ form an ON basis for $\mathrm{Ran}(\mathsf{Y}_\ell)$.)

Error measure:

The error incurred by the algorithm is $e_\ell = ||\mathsf{A} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}||$.

The error e_ℓ is bounded from below by $\sigma_{\ell+1} = \inf\{||\mathsf{A} - \mathsf{B}|| : \mathsf{B} \text{ has rank } \ell\}$.

Primitive problem: Given an $m \times n$ matrix A and an integer $\ell < \min(m, n)$, find an orthonormal $m \times \ell$ matrix Q such that $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

1. Construct a random matrix Ω_ℓ of size $n \times \ell$.
   Suppose for now that Ω_ℓ is Gaussian.

2. Form the $m \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.

3. Construct an $m \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.
   (In other words, the columns of Q_ℓ form an ON basis for $\mathrm{Ran}(\mathsf{Y}_\ell)$.)

Error measure:

The error incurred by the algorithm is $e_\ell = ||\mathsf{A} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}||$.

The error e_ℓ is bounded from below by $\sigma_{\ell+1} = \inf\{||\mathsf{A} - \mathsf{B}|| : \mathsf{B} \text{ has rank } \ell\}$.

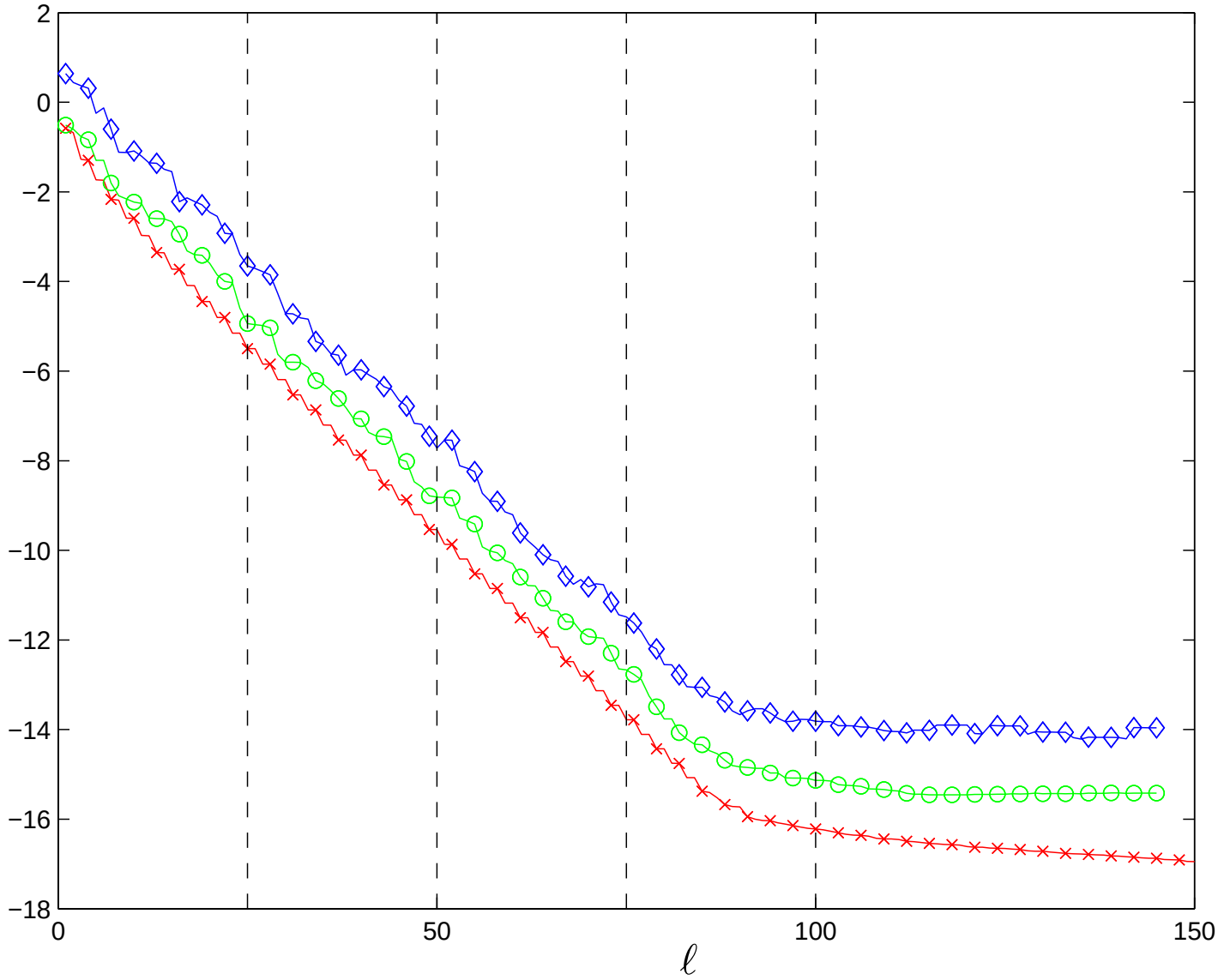
Error estimate: $f_\ell = \max_{1 \le j \le 10} \left|\left|\left(I - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\right)\boldsymbol{y}_{\ell+j}\right|\right|$.
The computation stops when we come to an ℓ such that $f_\ell < \varepsilon \times [\text{constant}]$.

Results from one realization of the randomized algorithm

$\log_{10}(10\, f_\ell)$
(error bound)

$\log_{10}(e_\ell)$
(actual error)

$\log_{10}(\sigma_{\ell+1})$
(theoretically
minimal error)



Note: The development of an error estimator resolves the issue of not knowing the numerical rank in advance!

Was this just a lucky realization?

No. In fact, the algorithm from a practical point of view behaves entirely like a deterministic method — you hardly see any difference at all from run to run.

For the incredulous, a rigorous theory, and plenty of numerical examples, are given in the following paper:

> *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions*
> N. Halko, P.G. Martinsson, J. Tropp — arXiv.org report 0909.4061.

Theorem: *[Halko, Martinsson, Tropp 2009] Fix a real $n \times n$ matrix A with singular values $\sigma_1, \sigma_2, \sigma_3, \ldots$. Choose integers $k \geq 1$ and $p \geq 2$, and draw an $n \times (k+p)$ standard Gaussian random matrix Ω. Construct the sample matrix $\mathsf{Y} = \mathsf{A}\Omega$, and let Q denote an orthonormal matrix such that $Ran(\mathsf{Q}) = Ran(\mathsf{Y})$. Then*

$$\mathbb{E}||\mathsf{A} - \mathsf{Q}\mathsf{Q}^*\mathsf{A}||_{\mathrm{F}} \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{1/2}.$$

Moreover,

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

Numerical experiments indicate that these estimates are close to sharp.

The paper also gives bounds on the tail probabilities; they turn out to *decay as p^{-p}.* For $p = 20$, the likelihood of "failure" is less than 10^{-17}.

Given an $m \times n$ matrix A of ε-rank k, compute an approximate rank-k SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Set $\ell = k + 10$.	(5) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Draw an $n \times \ell$ random matrix Ω.	(6) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Form the $m \times \ell$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(7) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.
(4) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(8) Truncate the last 10 terms.

Let us compare the scheme as described above to standard (non-randomized) methods in a few representative environments.

Given an $m \times n$ matrix A of ε-rank k, compute an approximate rank-k SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Set $\ell = k + 10$.	(5) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Draw an $n \times \ell$ random matrix Ω.	(6) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Form the $m \times \ell$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(7) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.
(4) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(8) Truncate the last 10 terms.

Assumption: A can rapidly be applied to vectors.
(Suppose A is sparse / FFT-able / an integral operator amenable to FMM / ...)

The randomized scheme should be compared to Krylov-methods such as Arnoldi and Lanzcos.

- The randomized method is more robust — error analysis does not depend on the spectrum of A, the method cannot be crippled by a poor starting vector, *etc.*
- The randomized method allows the ℓ matrix-vector products to be computed *concurrently*, this enables parallelization, out-of-core execution, BLAS3, *etc.*
- The simple randomized method stated above produces larger errors than Arnoldi, but there exist hybrid schemes (*e.g.* set $\mathsf{Y} = (\mathsf{A}\mathsf{A}^*)^q\,\mathsf{A}\,\Omega$) that combine the best properties of the two methods.

Given an $m \times n$ matrix A of ε-rank k, compute an approximate rank-k SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Set $\ell = k + 10$.	(5) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Draw an $n \times \ell$ random matrix Ω.	(6) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Form the $m \times \ell$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(7) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.
(4) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(8) Truncate the last 10 terms.

Assumption: A is presented as an array of numbers stored in fast memory (RAM).

Then replace the Gaussian random matrix by a *Subsampled Random Fourier Transform.* This can be applied in $O(mn \log(\ell))$ operations. Since $\ell \sim k$, we find

$$\text{complexity} \sim mn \log(k) + (m+n)k^2$$

Significant gains even for small matrices. Examples:

- $m = n = 2000$ and $k = 100$ gives a speed-up by a factor 4.5
- $m = n = 4000$ and $k = 200$ gives a speed-up by a factor 6.5

(The reference method was a highly optimized code using BLAS3, LAPACK, *etc.*)

Not my work (alas): [Liberty, Rokhlin, Tygert, Woolfe 2006]. Also [Ailon Chazelle 2006].

Given an $m \times n$ matrix A of ε-rank k, compute an approximate rank-k SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Set $\ell = k + 10$.	(5) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Draw an $n \times \ell$ random matrix Ω.	(6) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Form the $m \times \ell$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(7) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.
(4) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(8) Truncate the last 10 terms.

Assumption: A is presented as a huge array of numbers stored in slow memory.

The critical cost in this environment is the number of memory accesses that are needed. Flop count is largely irrelevant.

- The algorithm as stated requires only two passes over the data.
- With minor modifications, only a *single pass* is required.
  - This modified algorithm can execute on *streaming data.*
- Very few constraints on communication — excellent algorithm for parallel implementation.

Given an $m \times n$ matrix A of ε-rank k, compute an approximate rank-k SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Set $\ell = k + 10$.	(5) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Draw an $n \times \ell$ random matrix Ω.	(6) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Form the $m \times \ell$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(7) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.
(4) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(8) Truncate the last 10 terms.

Assumption: A is presented as a huge array of numbers stored in slow memory.

Problem: The huge matrices we would like to handle (image processing, PCA of stochastic data, etc) are typically very noisy in the sense that their singular values decay slowly. This renders the algorithm above very inaccurate.

Remedy: Introduce ideas from Krylov methods, for instance, set

$$\mathsf{Y} = (\mathsf{A}\mathsf{A}^*)^q \, \mathsf{A} \, \Omega.$$

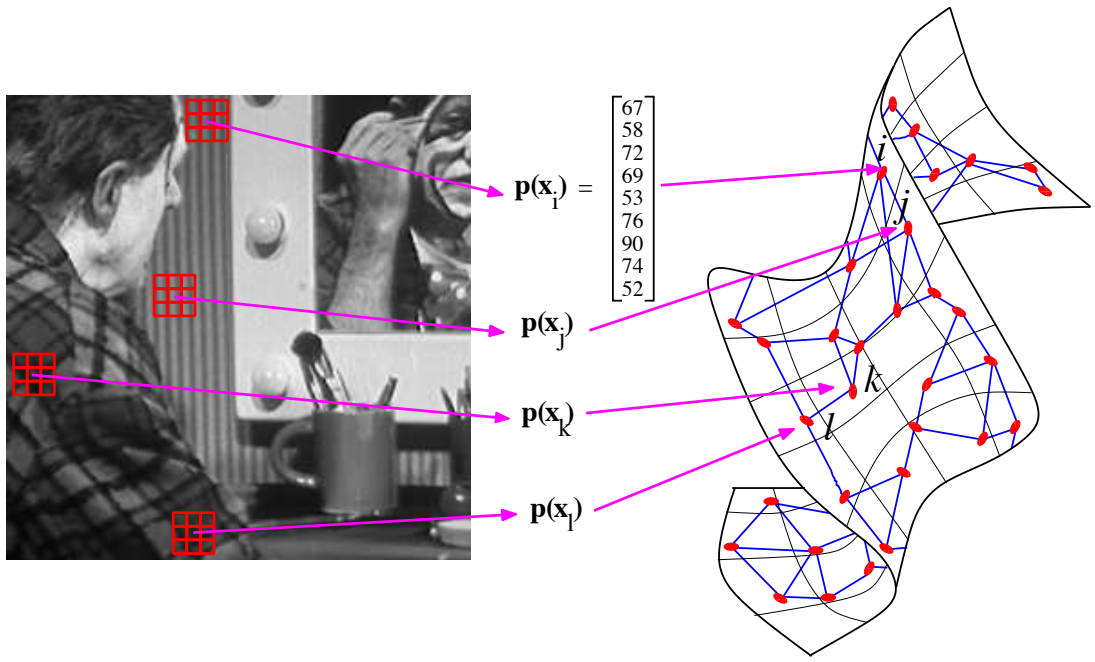
This magnifies the relative weight of the leading singular vectors at the price of requiring $2q + 1$ passes. However, $q = 2$ or $q = 3$ is often sufficient.

The resulting method allows the computation of the PCA of a huge (say $200\,000 \times 200\,000$) and very noisy matrix on a laptop.

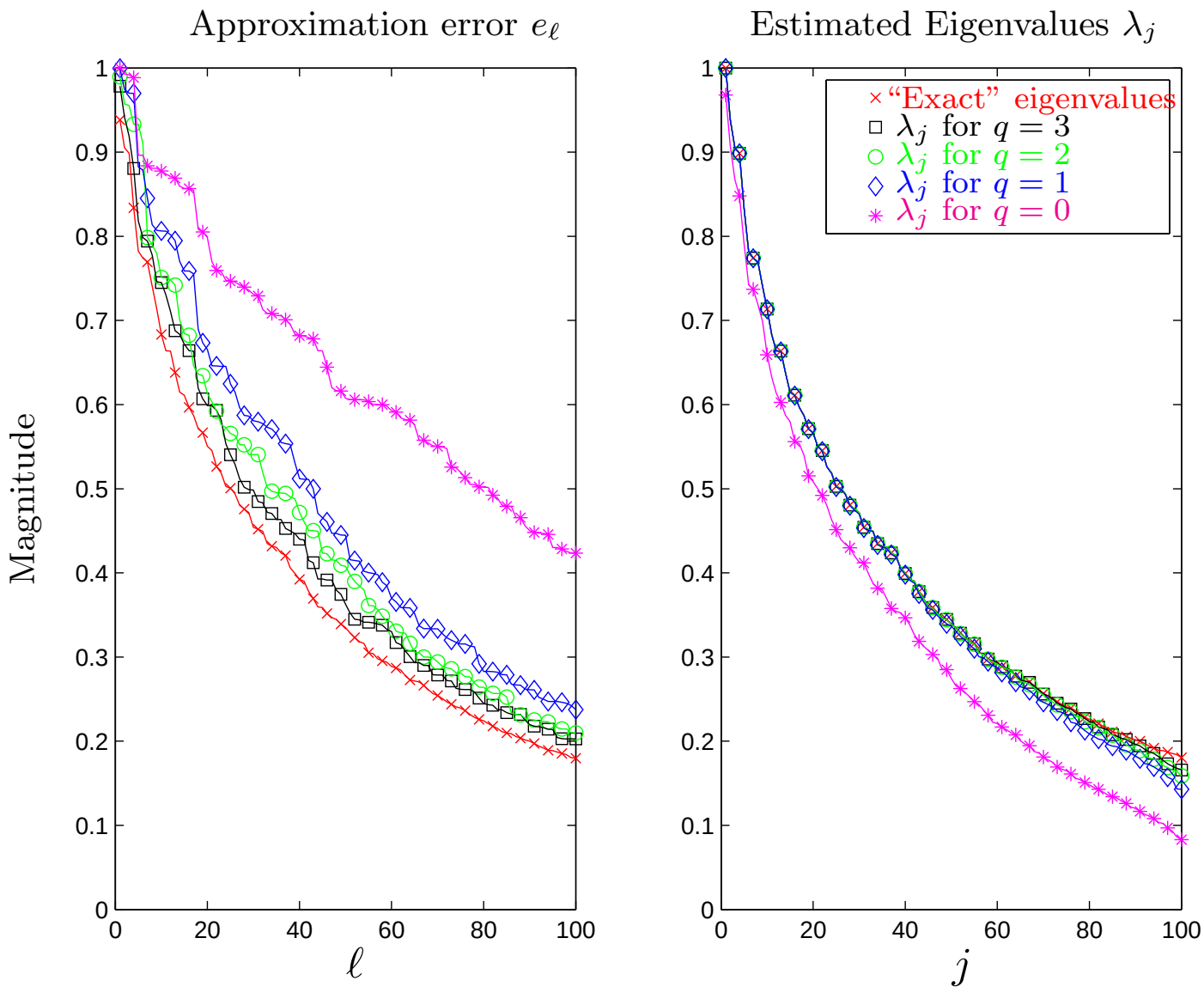
Example 1:

The matrix A being analyzed is a 9025×9025 matrix arising in a diffusion geometry approach to image processing.

To be precise, A is a graph Laplacian on the manifold of 9×9 patches.



Joint work with François Meyer of the University of Colorado at Boulder.



The pink lines illustrates the performance of the basic random sampling scheme. The errors are huge, and the estimated eigenvalues are much too small.

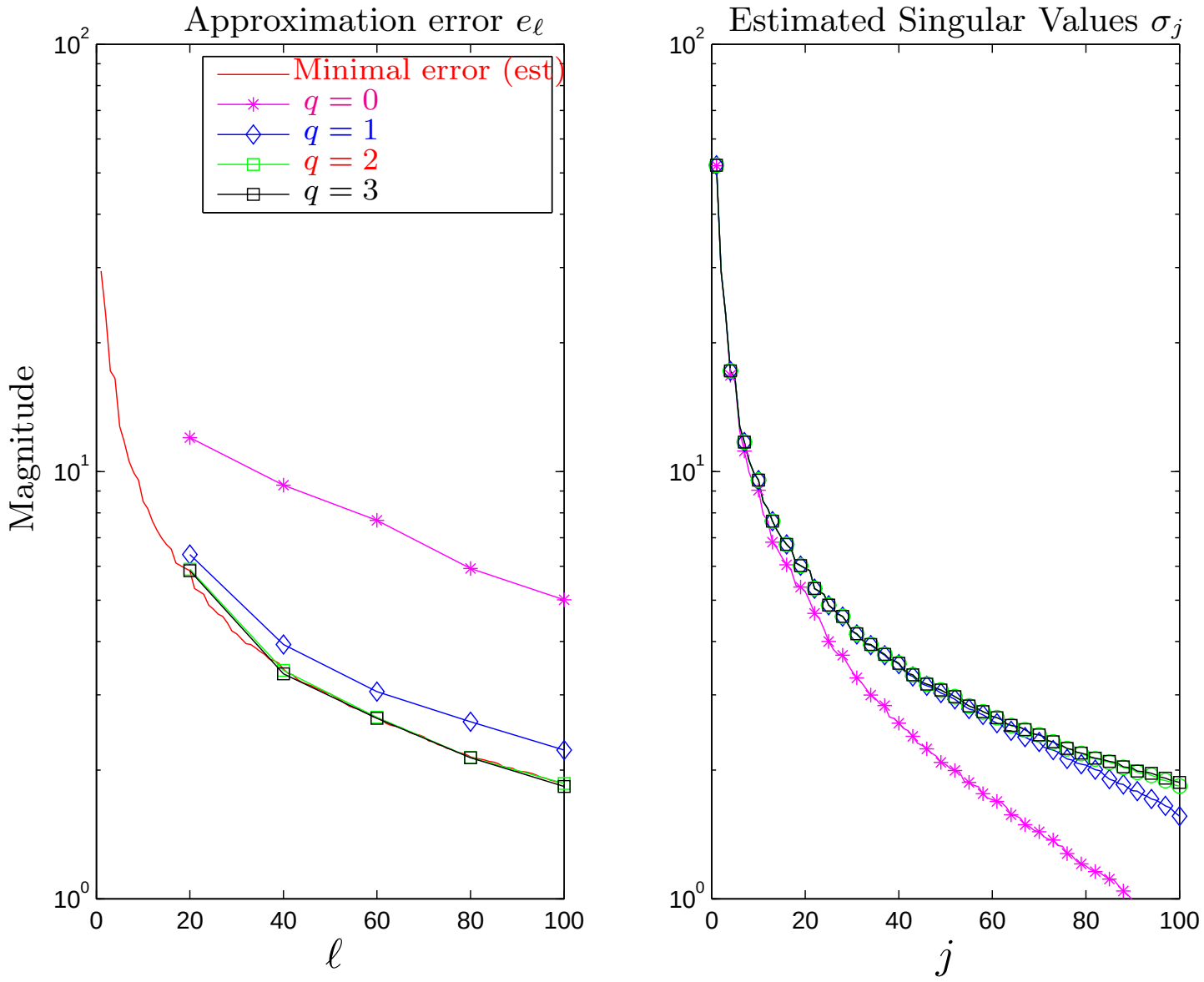
Example 2: "Eigenfaces"

We next process a data base containing $m = 7\,254$ pictures of faces

Each image consists of $n = 384 \times 256 = 98\,304$ gray scale pixels.

We center and scale the pixels in each image, and let the resulting values form a column of a $98\,304 \times 7\,254$ data matrix A.

The left singular vectors of A are the so called *eigenfaces* of the data base.



The pink lines illustrates the performance of the basic random sampling scheme. Again, the errors are huge, and the estimated eigenvalues are much too small.

Numerical examples

The computational examples are assembled to illustrate the *asymptotic scaling* of the methods.

Most of the examples are old: They were generated around 2005 on a PC from 2002 (a 2.8Ghz machine with 512Mb of RAM).

Other examples are more recent, these were implemented in Matlab.

Note: Still more bark than bite, but this will change ...

Example: An exterior Laplace Dirichlet problem

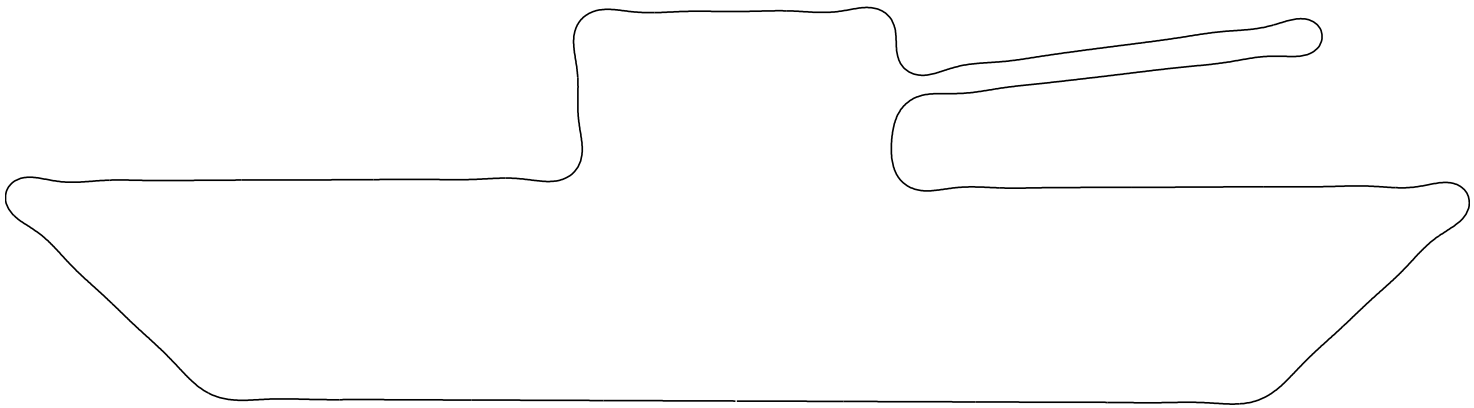
We invert a matrix approximating the operator

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

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

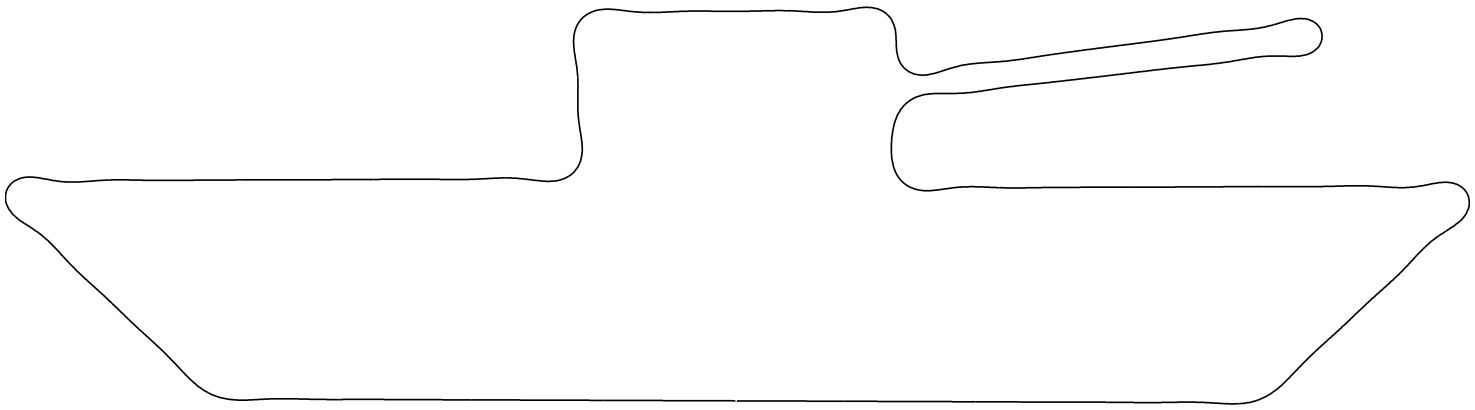
$$D(\boldsymbol{x},\boldsymbol{y}) = -\frac{1}{2\pi}\,\frac{\boldsymbol{n}(\boldsymbol{y})\cdot(\boldsymbol{x}-\boldsymbol{y})}{|\boldsymbol{x}-\boldsymbol{y}|^2},$$

and where Γ is the countour:



N_{start}	N_{final}	t_{tot} (sec)	t_{solve} (sec)	E_{res}	E_{pot}	σ_{min}	Memory (MB)
400	301	5.3e-01	2.9e-03	4.7e-10	3.0e-06	1.3e-02	4.2e+00
800	351	9.6e-01	4.1e-03	2.2e-10	6.3e-10	1.2e-02	6.5e+00
1600	391	1.6e+00	6.3e-03	1.3e-10	1.6e-10	1.2e-02	9.2e+00
3200	391	1.8e+00	8.5e-03	6.6e-11	3.7e-10	1.2e-02	1.1e+01
6400	391	2.2e+00	1.2e-02	5.9e-11	8.9e-11	1.2e-02	1.4e+01
12800	390	2.6e+00	1.9e-02	3.6e-11	5.9e-11	1.2e-02	2.1e+01
25600	391	3.9e+00	3.4e-02	2.7e-11	4.7e-10	—	3.5e+01
51200	393	6.5e+00	6.5e-02	2.5e-11	5.3e-11	—	6.3e+01
102400	402	1.3e+01	1.2e-01	2.0e-11	—	—	1.2e+02

Example: An exterior Helmholtz Dirichlet problem



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

The "combined field formulation" is used in forming the BIE.

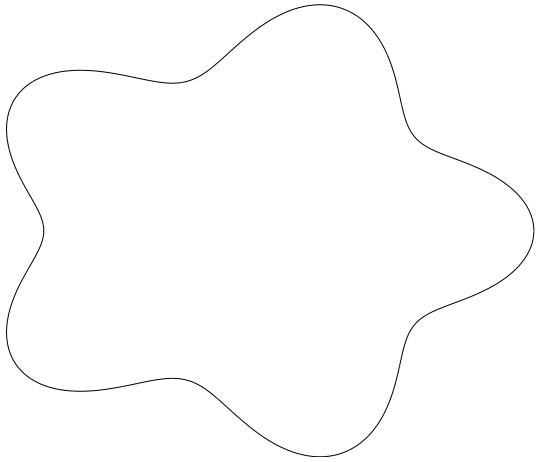
k	N_{start}	N_{final}	t_{tot} (sec)	t_{solve} (sec)	E_{res}	E_{pot}	σ_{min}	M (kB)
21	800	435	1.5e+01	3.3e-02	9.7e-08	7.1e-07	6.5e-01	12758
40	1600	550	3.0e+01	6.7e-02	6.2e-08	4.0e-08	8.0e-01	25372
79	3200	683	5.3e+01	1.2e-01	5.3e-08	3.8e-08	3.4e-01	44993
158	6400	870	9.2e+01	2.0e-01	3.9e-08	2.9e-08	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 $O(n + k^3)$.

(Corresponding Laplace problems are *much* faster,
inversion of a $10^5 \times 10^5$ matrix takes less than 20 seconds.)

Example: An interior Helmholtz Dirichlet problem

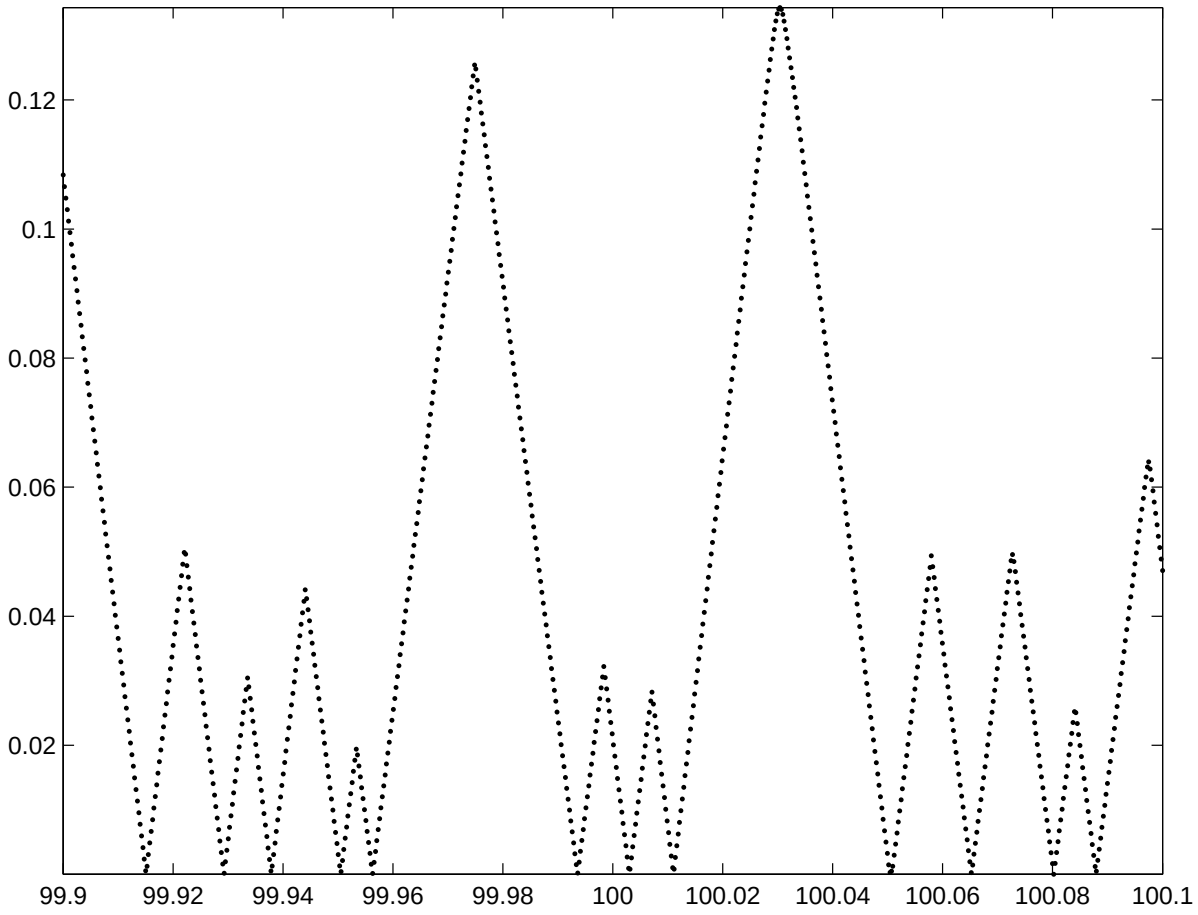


The diameter of the contour is about 2.5. An interior Helmholtz problem with Dirichlet boundary data was solved using $N = 6\,400$ discretization points, with a prescribed accuracy of 10^{-10}.

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}.



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.

What about finite element matrices?

These look quite different — *very* large, sparse, ...

However, their *inverses* have the rank structure of discretized integral operators.

Example: Consider the Laplace BVP

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

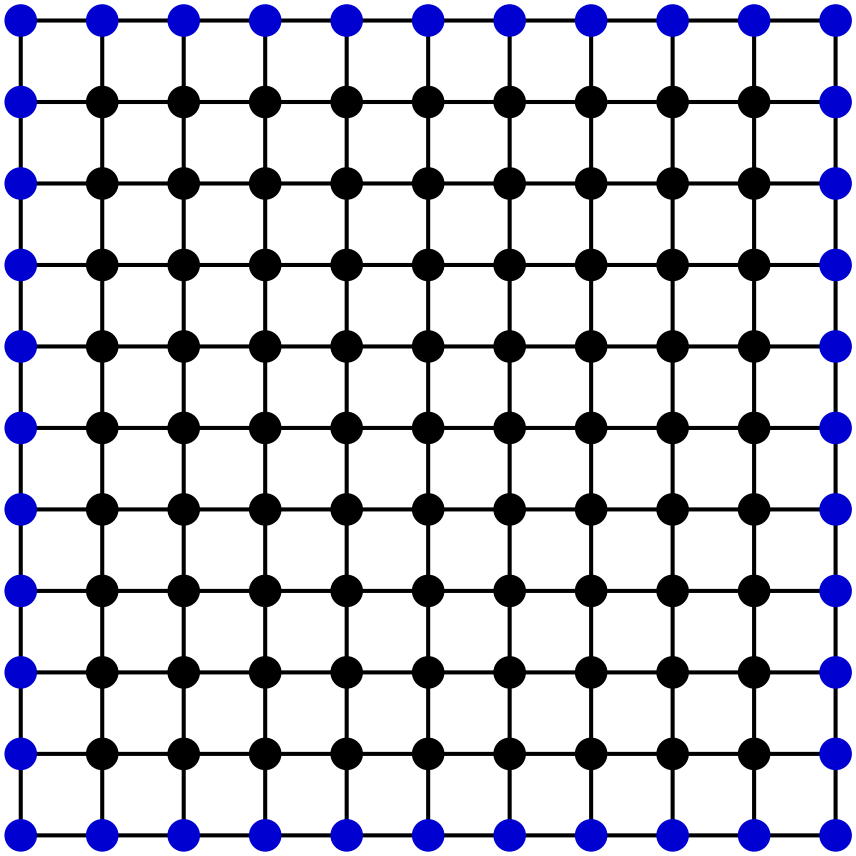
The finite element method produces a large sparse matrix A whose action mimics the action of the differential operator $-\Delta$.

The *inverse of* A mimics the action of the inverse operator

$$u(\boldsymbol{x}) = \int_\Omega G(\boldsymbol{x}, \boldsymbol{y})\, f(\boldsymbol{y})\, dA(\boldsymbol{y}),$$

where G is the Green's function of the problem.
(Note that G is know analytically only for the most trivial domains Ω.)

Example: 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:

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

N	T_{solve} (sec)	T_{apply} (sec)	M (kB)	e_1	e_2	e_3	e_4
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	—	—	—	—

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

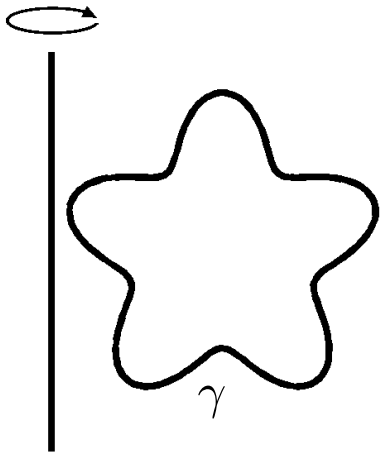
e_1 The largest error in any entry of $\tilde{\mathsf{A}}_n^{-1}$

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

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

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

Example: BIEs on rotationally symmetric surfaces



Generating curve

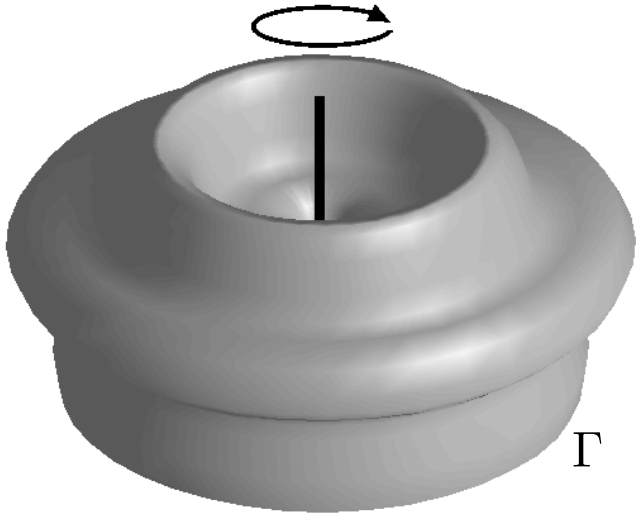
Let Γ be a surface of rotation generated by a curve γ, and consider a BIE associated with Laplace's equation:

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

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

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

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



Surface

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

$$N = 320\,000.$$

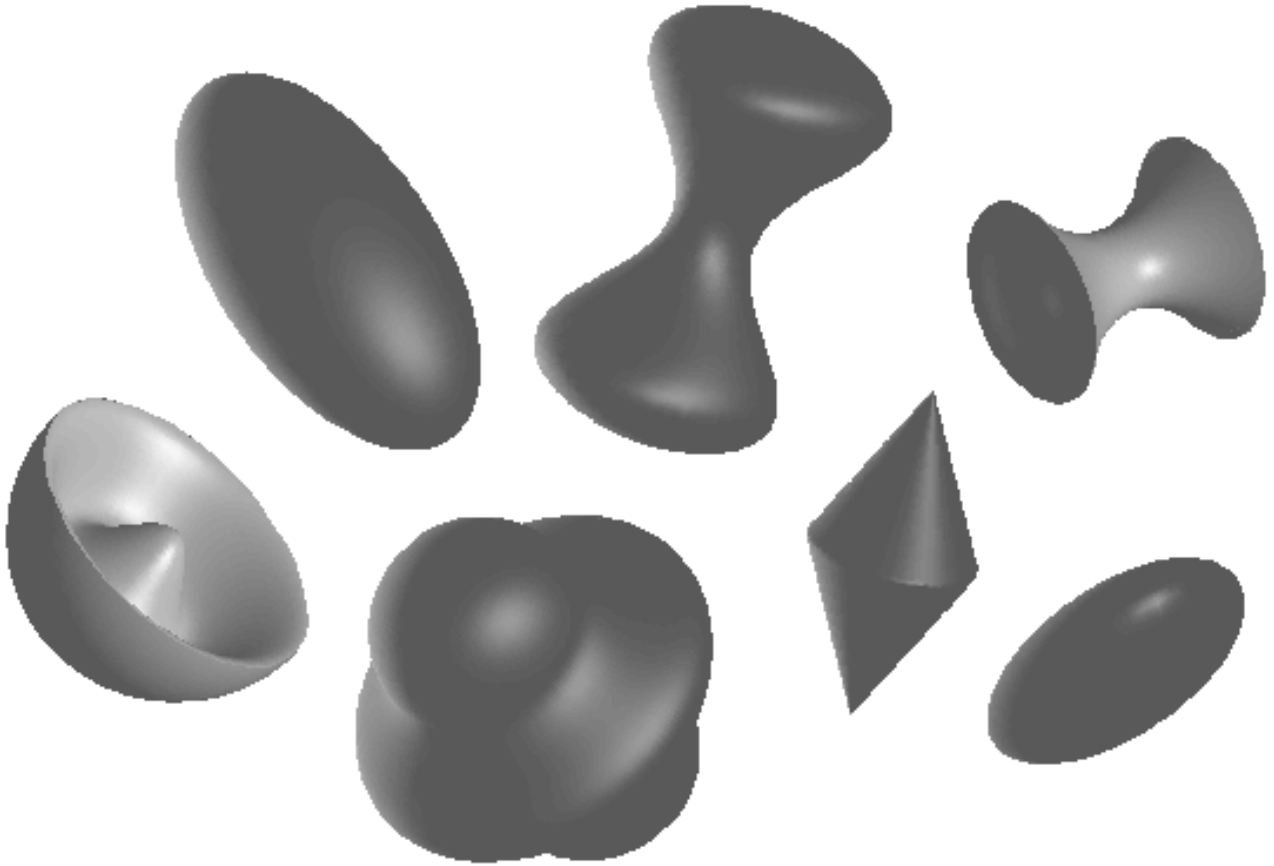
For typical loads, the relative error was less than 10^{-10} and the CPU times were

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

Work in progress: Extension to multibody acoustic scattering:

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

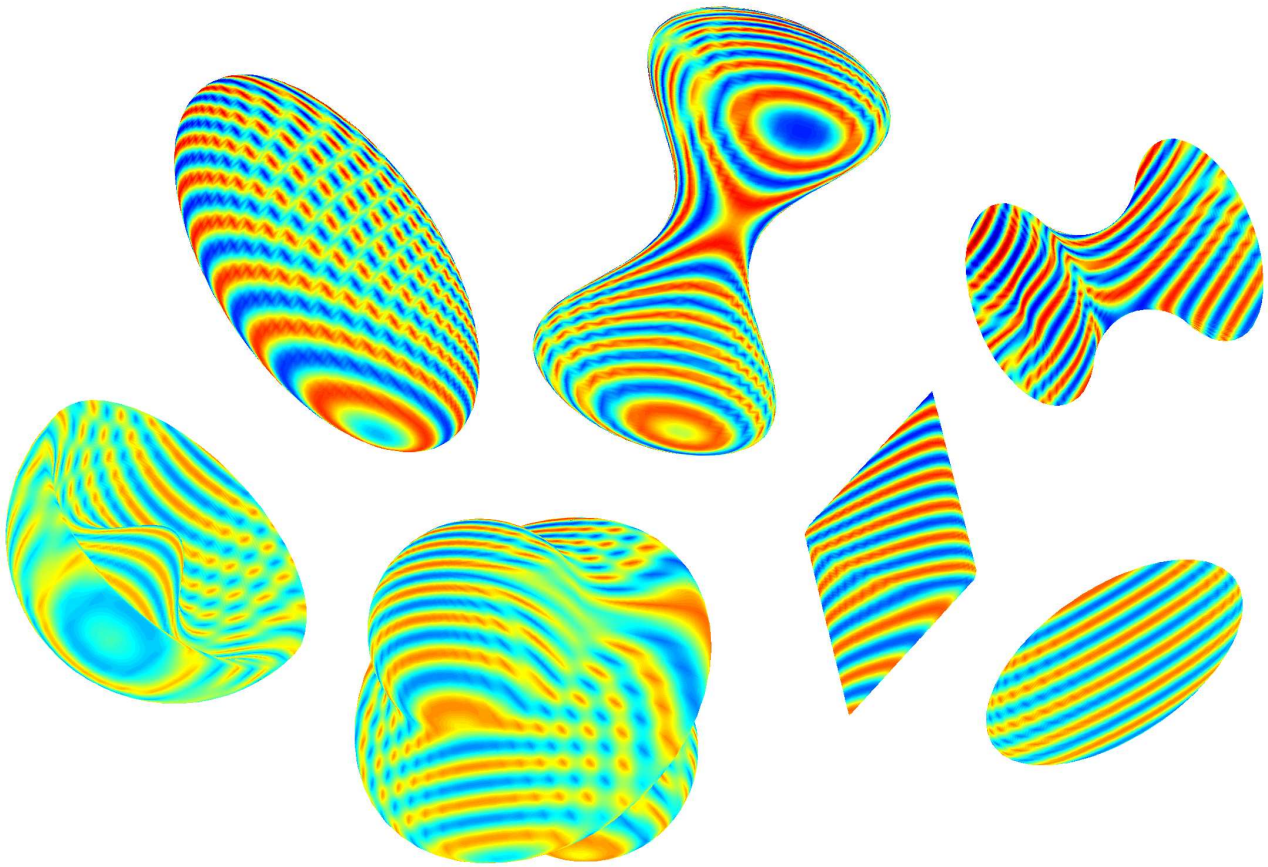
Inter-body interactions are handled via the wideband FMM and an iterative solver (GMRES appears to be working well).



Work in progress: 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 (GMRES appears to be working well).



3D problems from Denis.

Computation carried out by Denis Gueyffier at Courant.

Taken from Greengard, Gueyffier, Martinsson, Rokhlin, "Fast direct solvers for integral equations in complex three-dimensional domains", Acta Numerica 2009.

Assertions:

- Fast direct solvers excel for problems on 1D domains:
  - Integral operators on the line.
  - Boundary Integral Equations in $\mathbb{R}^2$.

  May well (should!) become standard solution technique in these environments.
- Fast direct solvers for large sparse "finite element matrices" associated with elliptic PDEs in $\mathbb{R}^2$ work well. Very competitive in certain environments.

Predictions:

- Very efficient and versatile fast direct methods will be developed for BIEs associated with non-oscillatory problems on surfaces in $\mathbb{R}^3$.
- *Randomized methods* will prove enormously helpful. They have already demonstrated their worth in large scale linear algebra.
- Direct solvers for scattering problems will be widely used, even with $O(N^{1.5})$ or $O(N^2)$ scaling. They parallelize very well, can use distributed memory, *etc.*

Open questions:

- Will direct solvers be competitive for volume problems in $\mathbb{R}^3$?
- Are $O(N)$ direct solvers for highly oscillatory problems possible?