Fast numerical methods for solving linear PDEs

Gunnar Martinsson
The University of Colorado at Boulder

Ph.D. Students:

Adrianna Gillman

Nathan Halko

Patrick Young

Collaborators:

Edo Liberty (Yale)

Vladimir Rokhlin (Yale)

Yoel Shkolnisky (Yale)

Joel Tropp (Caltech)

Mark Tygert (UCLA/Courant)

Franco Woolfe (Goldman Sachs)

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

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

  by a linear $N \times N$ system $A\,x = b$ 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.
  Such solvers are both "fast" and "direct".

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

$$\begin{array}{ccccc} A & = & B & & C \\ m \times n & & m \times k & & k \times n \end{array}$$

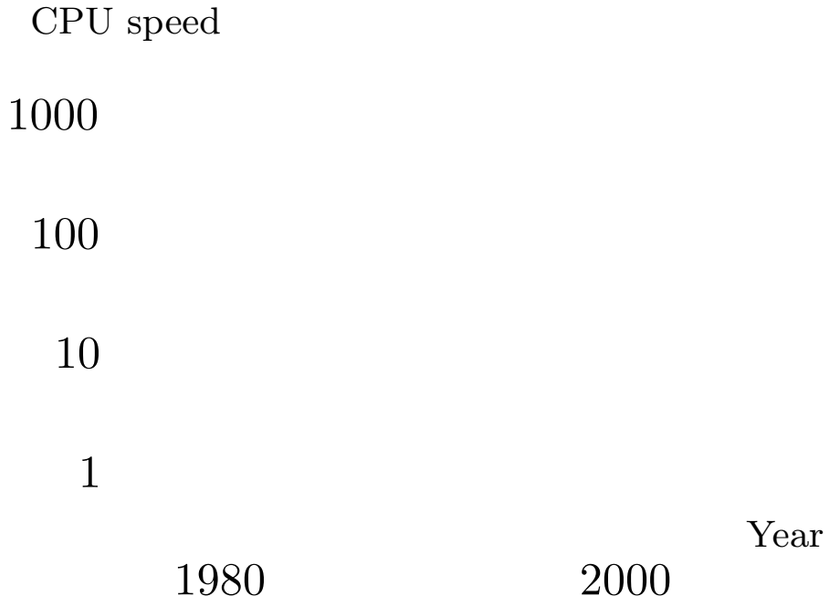
  New: Randomized methods that do this in $O(m\,n\,\log(k))$ operations.

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 $O(N(\log N)^2)$ are also called "fast".

Growth of computing power and the importance of algorithms



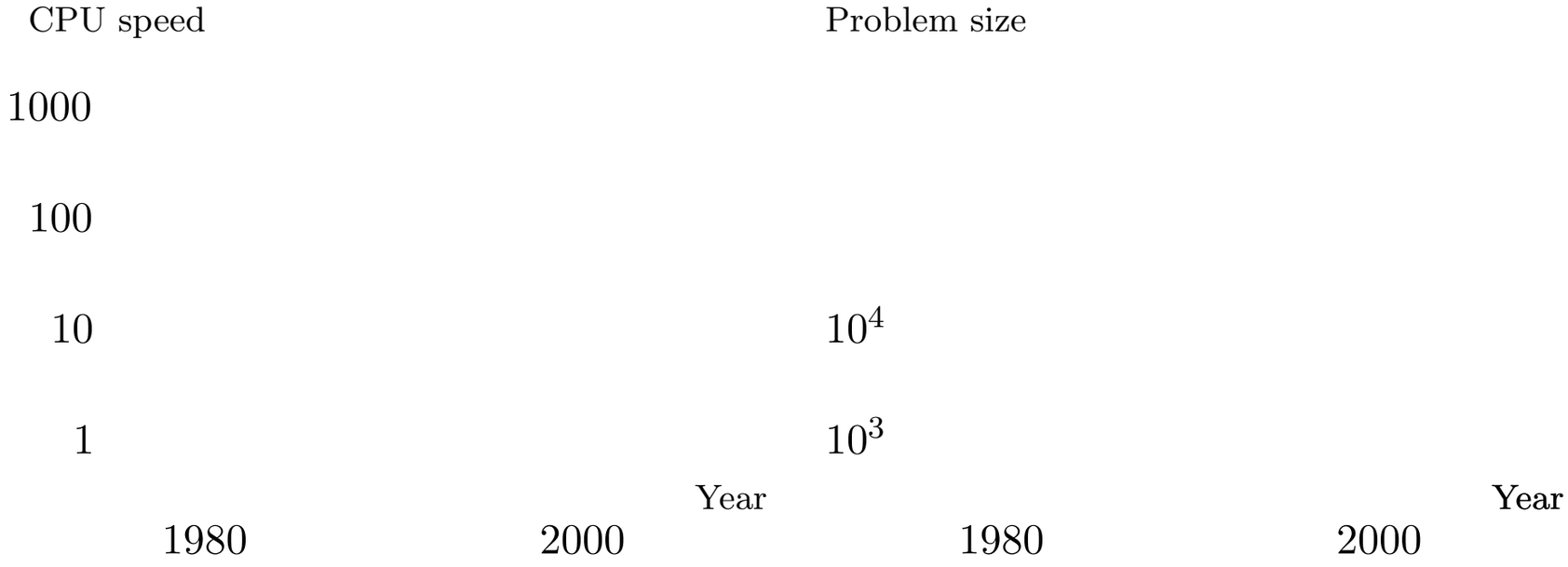
Consider the computational task of solving a linear system $A\,x = b$ of N algebraic equations with N unknowns.

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.*

Growth of computing power and the importance of algorithms



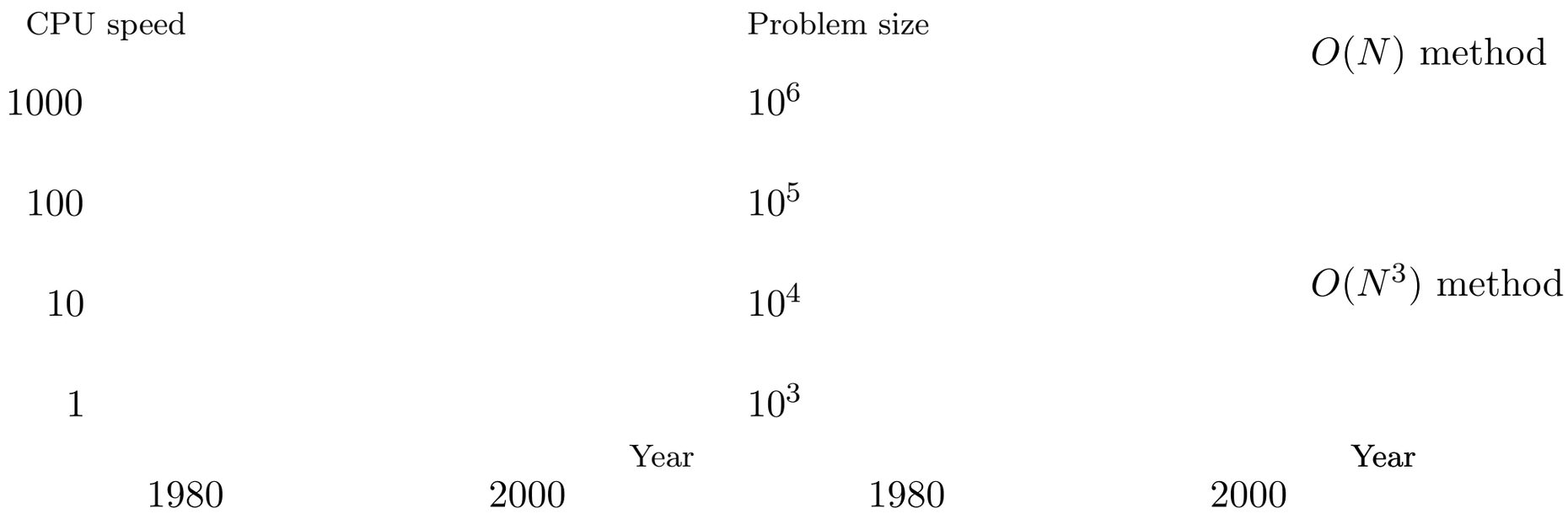
Consider the computational task of solving a linear system $A\,x = b$ of N algebraic equations with N unknowns.

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.*

Growth of computing power and the importance of algorithms



Consider the computational task of solving a linear system $A\,x = b$ 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.

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

The methods of this talk all parallelize naturally.

"Iterative" versus "direct" solvers

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

$$A\,x = b.$$

Iterative methods:	**Direct methods:**
Examples: GMRES, conjugate gradients, Gauss-Seidel, *etc.*	Examples: Gaussian elimination, LU factorizations, matrix inversion, *etc.*
Construct a sequence of vectors x_1, x_2, x_3, ... that (hopefully!) converge to the exact solution.	Always give an answer. Deterministic.
Many iterative methods access A only via its action on vectors.	Robust. No convergence analysis.
Often require problem specific preconditioners.	Great for multiple right hand sides.
High performance when they work well. $O(N)$ solvers.	Have often been considered too slow for high performance computing.
	(Directly access elements or blocks of A.)
	(Exact except for rounding errors.)

Discretization of linear Boundary Value Problems

We consider stationary linear Boundary Value Problems of the form

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

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

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

Example: Laplace's equation with Dirichlet boundary data:

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

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.

Advantages of direct solvers over iterative solvers:

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

2. Problems that are relatively ill-conditioned:
   - Scattering problems at intermediate or high frequencies.
   - Ill-conditioning due to geometry (elongated domains, percolation, etc).
   - Ill-conditioning due to lazy handling of corners, cusps, *etc.*
   - Finite element and finite difference discretizations.

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

Advantages of direct solvers over iterative solvers, continued:

Perhaps most important: **Engineering considerations.**

Direct methods tend to be more robust than iterative ones.

This makes them more suitable for "black-box" implementations.

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

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

Brief review of previous work on fast direct solvers:

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.*

What is "new" in our work:

- True $O(N)$ complexity (no log-factors).
- Algorithms that allow loops to be "un-nested".
- Small constants — these methods are fast not only in the technical sense.

How does it work?

As you would expect ... very technical ... quite involved notation ...

What follows is a 10 min description of the 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

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

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

(without the diagonal entries)

The 50^{th} row of A

(without the diagonal entries)

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

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

Singular values of A_{12} (now for a 200×200 matrix A):

$\log_{10}(\sigma_j)$

j

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

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

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

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

where G is the Green's function of the problem.
(Note that Ω is a general domain so G is not known analytically!)

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

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

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

$$\begin{array}{ccccc} A_{ij} & = & U_i & \tilde{A}_{ij} & V_j^* \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

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 $A = \begin{bmatrix} D_{11} & U_1 \tilde{A}_{12} V_2^* & U_1 \tilde{A}_{13} V_3^* & U_1 \tilde{A}_{14} V_4^* \\ U_2 \tilde{A}_{21} V_1^* & D_{22} & U_2 \tilde{A}_{23} V_3^* & U_2 \tilde{A}_{24} V_4^* \\ U_3 \tilde{A}_{31} V_1^* & U_3 \tilde{A}_{32} V_2^* & D_{33} & U_3 \tilde{A}_{34} V_4^* \\ U_4 \tilde{A}_{41} V_1^* & U_4 \tilde{A}_{42} V_2^* & U_4 \tilde{A}_{43} V_3^* & D_{44} \end{bmatrix}$.

Then A admits the factorization:

$$A = \underbrace{\begin{bmatrix} U_1 & & & \\ & U_2 & & \\ & & U_3 & \\ & & & U_4 \end{bmatrix}}_{=U} \underbrace{\begin{bmatrix} 0 & \tilde{A}_{12} & \tilde{A}_{13} & \tilde{A}_{14} \\ \tilde{A}_{21} & 0 & \tilde{A}_{23} & \tilde{A}_{24} \\ \tilde{A}_{31} & \tilde{A}_{32} & 0 & \tilde{A}_{34} \\ \tilde{A}_{41} & \tilde{A}_{42} & \tilde{A}_{43} & 0 \end{bmatrix}}_{=\tilde{A}} \underbrace{\begin{bmatrix} V_1^* & & & \\ & V_2^* & & \\ & & V_3^* & \\ & & & V_4^* \end{bmatrix}}_{=V^*} + \underbrace{\begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & D_3 & \\ & & & D_4 \end{bmatrix}}_{=D}$$

or

$$\underset{p\,n \times p\,n}{A} = \underset{p\,n \times p\,k}{U} \underset{p\,k \times p\,k}{\tilde{A}} \underset{p\,k \times p\,n}{V^*} + \underset{p\,n \times p\,n}{D},$$

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

$$\underset{N\times N}{A} = \underset{N\times K}{U}\ \underset{K\times K}{\tilde{A}}\ \underset{K\times N}{V^*} + \underset{N\times N}{D},$$

then

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

where

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

(provided all intermediate matrices are invertible).

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

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

then

$$\underset{pn \times pn}{A^{-1}} = \underset{pn \times pk}{E} \quad \underset{pk \times pk}{(\tilde{A} + \hat{D})^{-1}} \quad \underset{pk \times pn}{F^*} + \underset{pn \times pn}{G},$$

where

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

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!

We must recurse!

Using a telescoping factorization of A:

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

we have a formula

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

Block structure of factorization:

$$U^{(3)} \quad U^{(2)}\ U^{(1)}\ B^{(0)}\ (V^{(1)})^*\ B^{(1)}\ (V^{(2)})^* \qquad B^{(2)} \qquad\qquad (V^{(3)})^* \qquad\qquad\qquad D^{(3)}$$

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

Many important details are left out.

Among the more important ones:

- How do you represent potentials?
  Multipole expansions, proxy charges, interpolation, ...

- How do you compute the telescoping factorization in the first place?

- Generalization to
  - Volume integral equations in $\mathbb{R}^2$.
  - Boundary integral equations in $\mathbb{R}^3$.

  (Volume integral equations in $\mathbb{R}^3$ are currently not practical.)

Numerical examples

In developing direct solvers, the "proof is in the pudding" — recall that from a theoretical point of view, the problem was 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.

Example: 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}	$\sigma_{\min}$	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 $O(n + k^3)$.

(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.

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:

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

N	T_{invert} (seconds)	T_{apply} (seconds)	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	—	—	—	—

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 $\tilde{A}_{nn}^{-1}$.

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

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

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

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.

Key points — fast direct solvers

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

Future directions:

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

Part 2 — Approximation of matrices via randomized sampling

Notation:

A vector $x \in \mathbb{R}^n$ is measured using the ℓ^2 (Euclidean) norm:

$$||x|| = \left(\sum_{j=1}^{n} x_j^2 \right)^{1/2}.$$

A matrix $A \in \mathbb{R}^{m \times n}$ is measured using the corresponding operator norm

$$||A|| = \sup_{x \neq 0} \frac{||A\,x||}{||x||}.$$

Low-rank approximation

An $N \times N$ matrix A has rank k if there exist matrices B and C such that

$$\underset{N \times N}{A} = \underset{N \times k}{B} \; \underset{k \times N}{C}.$$

When $k \ll N$, computing the factors B and C is advantageous:

- Storing B and C require $O(N\,k)$ storage instead of $O(N^2)$.
- A matrix-vector multiply requires $2\,N\,k$ flops instead of N^2 flops.
- Certain factorizations reveal properties of the matrix.

In actual applications, we are typically faced with approximation problems:

Problem 1: Given a matrix A and a precision ε, find the minimal k such that

$$\min\{||A - \tilde{A}|| : \text{rank}(\tilde{A}) = k\} \leq \varepsilon.$$

Problem 2: Given a matrix A and an integer k, determine

$$A_k = \text{argmin}\{||A - \tilde{A}|| : \text{rank}(\tilde{A}) = k\}.$$

The singular value decomposition (SVD) provides the exact answer.

Any $m \times n$ matrix A admits a factorization (assuming $m \geq n$)

$$A = U\,D\,V^{\mathrm{t}} = [u_1\ u_2\ \cdots\ u_n] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_n \end{bmatrix} \begin{bmatrix} v_1^{\mathrm{t}} \\ v_2^{\mathrm{t}} \\ \vdots \\ v_n^{\mathrm{t}} \end{bmatrix} = \sum_{j=1}^{n} \sigma_j\, u_j\, v_j^{\mathrm{t}}.$$

σ_j is the j'th "singular value" of A
u_j is the j'th "left singular vector" of A
v_j is the j'th "right singular vector" of A.

Then:

$$\sigma_j = \inf_{\mathrm{rank}(\tilde{A})=j-1} ||A - \tilde{A}||.$$

and

$$\mathrm{argmin}\{||A - \tilde{A}|| :\ \mathrm{rank}(\tilde{A}) = k\} = \sum_{j=1}^{k} \sigma_j\, u_j\, v_j^{\mathrm{t}}.$$

The decay of the singular values determines how well a matrix can be approximated by low-rank factorizations.

Example: Let A be the 25×25 Hilbert matrix, *i.e.* $A_{ij} = 1/(i+j-1)$.
Let σ_j denote the j'th singular value of A.

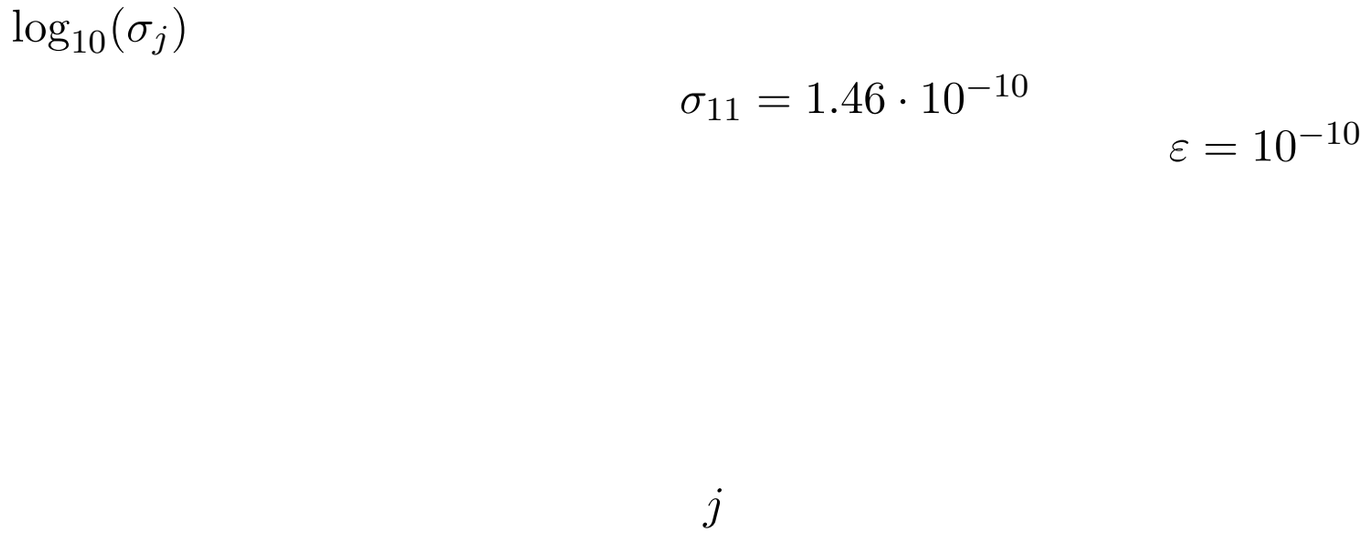
$$\log_{10}(\sigma_j)$$

$$j$$

For instance, to precision $\varepsilon = 10^{-10}$, the matrix A has rank 11.

The decay of the singular values determines how well a matrix can be approximated by low-rank factorizations.

Example: Let A be the 25×25 Hilbert matrix, *i.e.* $A_{ij} = 1/(i+j-1)$. Let σ_j denote the j'th singular value of A.



For instance, to precision $\varepsilon = 10^{-10}$, the matrix A has rank 11.

Model problem: Find an approximate basis for the column space of a given matrix.	• Let ε denote the computational accuracy desired. • Let A be an $N \times N$ matrix. • Determine an integer k and an $N \times k$ ON-matrix Q such that $\|\|A - Q\,Q^{\mathrm{t}}\,A\|\| \leq \varepsilon$.

Model problem: Find an approximate basis for the column space of a given matrix.	• Let ε denote the computational accuracy desired. • Let A be an $N \times N$ matrix. • Determine an integer k and an $N \times k$ ON-matrix Q such that $\|\|A - Q\,Q^{\mathrm{t}}\,A\|\| \leq \varepsilon$.

Notes:

- Once Q has been constructed, it is in many environments possible to construct standard factorization (such as the SVD/PCA) using $O(N\,k^2)$ operations. Specifically, this is true if either
  - matrix vector products $x \mapsto x'\,A$ can be evaluated rapidly, or,
  - individual entries of A can be computed in $O(1)$ operations.
- We seek a k that is as small as possible, but it is not a priority to make it absolutely optimal.
- If the Q initially constructed has too many columns, but is accurate, then the true optimal rank is revealed by postprocessing.

Model problem: Find an approximate basis for the column space of a given matrix.	• Let ε denote the computational accuracy desired. • Let A be an $N \times N$ matrix. • Determine an integer k and an $N \times k$ ON-matrix Q such that $\|\|A - Q\,Q^{\mathrm{t}}\,A\|\| \leq \varepsilon$.

We will discuss two environments:

Case 1:

We have a fast technique for evaluating matrix-vector products. Let T_{mult} denote the cost. We assume $T_{\text{mult}} \ll N^2$.

Standard methods (*e.g.* Lanczos) require $O(T_{\text{mult}}\,k)$ operations.

The new methods are also $O(T_{\text{mult}}\,k)$ but are more robust, more accurate, and better suited for parallelization.

Case 2:

A is a general $N \times N$ matrix.

Standard methods (*e.g.* Gram-Schmidt) require $O(N^2\,k)$ operations.

The new method requires $O(N^2\,\log(k))$ operations.

The methods that we propose are based on **randomized sampling.**

This means that they have a non-zero probability of giving an answer that is not accurate to within the requested accuracy.

The probability of failure can be balanced against computational cost by the user.

It can very cheaply be rendered entirely negligible; failure probabilities less than 10^{-15} are standard. (In other words, if you computed 1 000 matrix factorizations a second, you would expect to see one "failure" every 30 000 years.)

Definition: We say that an $m \times n$ matrix Ω is a Gaussian random matrix if

$$\Omega = \begin{bmatrix} \omega_{11} & \omega_{12} & \cdots & \omega_{1n} \\ \omega_{21} & \omega_{22} & \cdots & \omega_{2n} \\ \vdots & \vdots & & \vdots \\ \omega_{m1} & \omega_{m2} & \cdots & \omega_{mn} \end{bmatrix},$$

where the numbers ω_{ij} are random variables drawn independently from a normalized Gaussian distribution.

Note: The probability distribution of Ω is isotropic in the sense that if $U \in \mathrm{O}(m)$ and $V \in \mathrm{O}(n)$, then $U\,\Omega\,V$ has the same distribution as Ω.

Note: In practise, the random numbers used will be constructed using "random number generators." The quality of the generators will not matter much. (Shockingly little, in fact.)

We start with Case 1: We know how to compute the product $x \mapsto A\,x$ rapidly.

Algorithm 1: *Rapid computation of a low-rank approximation.*	• Let ε denote the computational accuracy desired. • Let A be an $N \times N$ matrix of ε-rank k. • We seek a basis for Col(A). • We can perform matrix-vector multiplies fast.

1. Fix a small positive integer p (representing how much "oversampling" we do). Construct a Gaussian random matrix Ω of size $n \times (k+p)$.

2. Form the $m \times (k+p)$ matrix $Y = A\,\Omega$.

3. Construct an $m \times (k+p)$ orthogonal matrix Q such that $Y = Q\,Q^{\mathrm{t}}\,Y$.

Each column of Y is a sample from the column space of A.
The more samples we have, the more likely it is that

$$||A - Q\,Q^{\mathrm{t}}\,A|| \leq \varepsilon. \tag{2}$$

If we were very lucky, then (2) would hold with $p = 0$.

Question: How large does p need to be in practice?

How to measure "how well we are doing":

Let Ω_ℓ be a Gaussian random matrix of size $n \times \ell$.

Set $Y_\ell = [y_1, y_2, \ldots, y_\ell] = A\,\Omega_\ell$.

Let Q_ℓ be an $m \times \ell$ matrix such that $Y_\ell = Q_\ell\, Q_\ell^{\mathrm{t}}\, Y_\ell$.

The "error" after ℓ steps is then

$$e_\ell = ||A - Q_\ell\, Q_\ell^{\mathrm{t}}\, A|| = ||(I - Q_\ell\, Q_\ell^{\mathrm{t}})\, A||.$$

The quantity e_ℓ should be compared to the minimal error

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

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

$$f_\ell = \max_{1 \le j \le 10} \big|\big|\big(I - Q_\ell\, Q_\ell^{\mathrm{t}}\big)\, y_{l+j}\big|\big|.$$

The computation stops when we come to an ℓ such that $f_\ell < \varepsilon \times$ [constant].

Specific example to illustrate the performance:
Let A be a 200×200 matrix arising from discretization of

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

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

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

Results from one realization of the randomized algorithm

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

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

ℓ

How to measure "how well we are doing" — revisited:

Let Ω_ℓ be a Gaussian random matrix of size $n \times \ell$.

Set $Y_\ell = [y_1, y_2, \ldots, y_\ell] = A\,\Omega_\ell$.

Let Q_ℓ be an $m \times \ell$ matrix such that $Y_\ell = Q_\ell\, Q_\ell^{\mathrm{t}}\, Y_\ell$.

The "error" after ℓ steps is then

$$e_\ell = ||A - Q_\ell\, Q_\ell^{\mathrm{t}}\, A|| = ||(I - Q_\ell\, Q_\ell^{\mathrm{t}})\, A||.$$

The quantity e_ℓ should be compared to the minimal error

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

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

$$f_\ell = \max_{1 \le j \le 10} \big|\big|\big(I - Q_\ell\, Q_\ell^{\mathrm{t}}\big)\, y_{l+j}\big|\big|.$$

The computation stops when we come to an ℓ such that $f_\ell < \varepsilon \times$ [constant].

How to measure "how well we are doing" — revisited:

Let Ω_ℓ be a Gaussian random matrix of size $n \times \ell$.

Set $Y_\ell = [y_1, y_2, \ldots, y_\ell] = A\,\Omega_\ell$.

Let Q_ℓ be an $m \times \ell$ matrix such that $Y_\ell = Q_\ell\, Q_\ell^{\mathrm{t}}\, Y_\ell$.

The "error" after ℓ steps is then

$$e_\ell = ||A - Q_\ell\, Q_\ell^{\mathrm{t}}\, A|| = ||(I - Q_\ell\, Q_\ell^{\mathrm{t}})\, A||.$$

The quantity e_ℓ should be compared to the minimal error

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

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

$$f_\ell = \max_{1 \le j \le 10} \big|\big|\big(I - Q_\ell\, Q_\ell^{\mathrm{t}}\big)\, y_{l+j}\big|\big|.$$

The computation stops when we come to an ℓ such that $f_\ell < \varepsilon \times$ [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)

ℓ

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

Was this just a lucky realization?

Was this just a lucky realization?

Results from 2000 realizations:

Important: What is stochastic is the *run time*, not the accuracy;
the error in the factorization was *always* less than 10^{-10},
and post-processing correctly determined the rank *every time*.

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

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

(error bound)

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

(actual error)

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

(theoretically

minimal error)

ℓ

So far, we have assumed that we have a fast matrix-vector multiplier at our disposal.

What happens if we do not?

In this case, $T_{\text{mult}} = N^2$ so the computational cost of Algorithm I is

$$O(T_{\text{mult}}\, k + N\, k^2) = O(N^2\, k + N\, k^2).$$

When $k \ll N$, Algorithm 1 might be slightly faster than Gram-Schmidt:

Multiplications required for Algorithm 1:	$N^2\,(k+10)$	$+O(k^2\,N)$
Multiplications required for Gram-Schmidt:	$N^2\,2\,k$	

Other benefits (sometimes more important ones than CPU count):

- Data-movement.
- Parallelization.
- More accurate. (This requires some additional twists not yet described.)

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

So far, we have assumed that we have a fast matrix-vector multiplier at our disposal.

What happens if we do not?

In this case, $T_{\text{mult}} = N^2$ so the computational cost of Algorithm I is

$$O(T_{\text{mult}}\, k + N\, k^2) = O(N^2\, k + N\, k^2).$$

When $k \ll N$, Algorithm 1 might be slightly faster than Gram-Schmidt:

Multiplications required for Algorithm 1:	$N^2\,(k+10)$	$+O(k^2\,N)$
Multiplications required for Gram-Schmidt:	$N^2\,2\,k$	

Other benefits (sometimes more important ones than CPU count):

- Data-movement.
- Parallelization.
- More accurate. (This requires some additional twists not yet described.)

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

Algorithm 2: An $O(N^2 \log(k))$ algorithm for *general* matrices:

Proposed by Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert.

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

$$\begin{array}{ccccc} Y & = & A & & \Omega. \\ N \times \ell & & N \times N & & N \times \ell \end{array}$$

Key points:

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

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

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

$$a \mapsto a\,\Omega$$

can be evaluated using $N \log(\ell)$ operations.

What is this "random but structured" matrix Ω?

$$\begin{array}{ccccc} \Omega & = & D & F & S \\ N \times \ell & & N \times N & N \times N & N \times \ell \end{array}$$

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 Ω 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 on square matrices of various sizes

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

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.

The accuracy of the randomized method has recently been improved.

Theory / context

In the remainder of the talk we will focus on Algorithm 1 (for the case when fast matrix-vector multiplies are available). For this case, we have fairly sharp estimates of the "failure probabilities."

The theoretical results to be presented are related to (and in some cases inspired by) earlier work on randomized methods in linear algebra. This work includes:

C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala (2000)

A. Frieze, R. Kannan, and S. Vempala (1999, 2004)

D. Achlioptas and F. McSherry (2001)

P. Drineas, R. Kannan, M. W. Mahoney, and S. Muthukrishnan (2006a, 2006b, 2006c, 2006d)

S. Har-Peled (2006)

A. Deshpande and S. Vempala (2006)

S. Friedland, M. Kaveh, A. Niknejad, and H. Zare (2006)

T. Sarlós (2006a, 2006b, 2006c)

Theorem (Martinsson, Rokhlin, Tygert 2006):

Let A be an $M \times N$ matrix.

Let k and p be positive integers. (k is "rank" and p is the degree of "oversampling")

Let Ω be an $N \times (k+p)$ Gaussian random matrix.

Let Q be an $M \times (k+p)$ matrix whose columns form an ON-basis for the columns of $A\,\Omega$.

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

Then

$$||A - Q\,Q^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{(k+p)\,(N-k)}\;\sigma_{k+1}, \qquad \leftarrow \textit{Not good!}$$

with probability at least

$$1 - \varphi(p),$$

where φ is a decreasing function satisfying, e.g.,

$$\varphi(5) < 3 \cdot 10^{-6}, \qquad \varphi(10) < 3 \cdot 10^{-13}, \qquad \varphi(15) < 8 \cdot 10^{-21}, \qquad \varphi(20) < 6 \cdot 10^{-27}.$$

Theorem (Martinsson, Rokhlin, Tygert 2006):

Let A be an $M \times N$ matrix.

Let k and p be positive integers. (k is "rank" and p is the degree of "oversampling")

Let Ω be an $N \times (k+p)$ Gaussian random matrix.

Let Q be an $M \times (k+p)$ matrix whose columns form an ON-basis for the columns of $A\,\Omega$.

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

Then

$$||A - Q\,Q^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{(k+p)\,(N-k)}\;\sigma_{k+1}, \qquad \leftarrow \textit{Not good!}$$

with probability at least

$$1 - \varphi(p),$$

where φ is a decreasing function satisfying, e.g.,

$$\varphi(5) < 3 \cdot 10^{-6}, \qquad \varphi(10) < 3 \cdot 10^{-13}, \qquad \varphi(15) < 8 \cdot 10^{-21}, \qquad \varphi(20) < 6 \cdot 10^{-27}.$$

The key bound in the proof is the line:

$$||A - Q\,Q^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{(k+p)\,(N-k)}\;\sigma_{k+1}.$$

The factor in blue represents the degree of suboptimality.

In applications where the singular values decay rapidly, this factor does not represent a problem. (A slight increase in k kills off the factor.)

The key bound in the proof is the line:

$$||A - Q\,Q^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{(k+p)\,(N-k)}\,\sigma_{k+1}.$$

The factor in blue represents the degree of suboptimality.

In applications where the singular values decay rapidly, this factor does not represent a problem. (A slight increase in k kills off the factor.)

However, the applications described at the beginning of the talk are very noisy, and it may be that

$$\sigma_{k+1} \approx \sigma_{k+2} \approx \cdots \approx \sigma_N \approx 10^{-2} \times \sigma_1.$$

The key bound in the proof is the line:

$$||A - Q\,Q^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{(k+p)\,(N-k)}\,\sigma_{k+1}.$$

The factor in blue represents the degree of suboptimality.

In applications where the singular values decay rapidly, this factor does not represent a problem. (A slight increase in k kills off the factor.)

However, the applications described at the beginning of the talk are very noisy, and it may be that

$$\sigma_{k+1} \approx \sigma_{k+2} \approx \cdots \approx \sigma_N \approx 10^{-2} \times \sigma_1.$$

Moreover, N may be very large. $N \sim 10^5$ would be common.

The key bound in the proof is the line:

$$||A - Q\,Q^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{(k+p)\,(N-k)}\,\sigma_{k+1}.$$

The factor in blue represents the degree of suboptimality.

In applications where the singular values decay rapidly, this factor does not represent a problem. (A slight increase in k kills off the factor.)

However, the applications described at the beginning of the talk are very noisy, and it may be that

$$\sigma_{k+1} \approx \sigma_{k+2} \approx \cdots \approx \sigma_N \approx 10^{-2} \times \sigma_1.$$

Moreover, N may be very large. $N \sim 10^5$ would be common.

The suboptimality is damning in data mining and signal processing applications. Here we have HUGE matrices, and lots of noise.

Theorem: *[Halko, Martinsson, Tropp 2009] Fix a* real $m \times n$ *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 test matrix* Ω. *Construct the data matrix* $Y = A\Omega$, *and let* P_A *denote the orthogonal projection onto the range of* Y. *Then*

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

Moreover,

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

- Numerical experiments indicate that these estimates are close to sharp.
- When $\sigma_j \sim \beta^j$, we have $\left(\sum_{j=k+1}^{\infty} \sigma_j^2\right)^{1/2} \sim \sigma_{k+1} \frac{1}{1-\beta}$.
- Tail probabilities are often in a practical sense irrelevant.

Power method for improving accuracy:

Note that the error depends on how quickly the singular values decay.

The faster the singular values decay — the higher the relative weight of the dominant modes are weighted in the samples.

Idea: The matrix $B = (A\,A^*)^q\,A$ has the same left singular vectors as A, and its singular values are

$$\sigma_j(B) = (\sigma_j(A))^{2\,q+1}.$$

Much faster decay — so use the sample matrix

$$Z = B\,\Omega = (A\,A^*)^q\,A\,\Omega$$

instead of

$$Y = A\,\Omega.$$

Power method for improving accuracy:

The following theorem is inspired by results by Rokhlin, Szlam, and Tygert (2008):

Theorem: *[Halko, Martinsson, Tropp 2009] Let m, n, and ℓ be positive integers such that $\ell < n \leq m$. Let A be an $m \times n$ matrix and let Ω be an $n \times \ell$ matrix. Let q be a non-negative integer, set $B = (A^* A)^q A$, and construct the sample matrix $Z = B\,\Omega$. Then*

$$|||(I - P_Z)\,A||| \leq |||(I - P_Z)\,B|||^{1/(2q+1)}$$

where $|||\cdot|||$ denotes either the spectral norm or the Frobenius norm.

Since the ℓ'th singular value of $B = (A^* A)^q A$ is $\sigma_l^{2\,q+1}$, any result of the type

$$||(I - P_Y)\,A|| \leq C\,\sigma_{k+1},$$

where $Y = A\,\Omega$ and $C = C(m, n, k)$, gets improved to a result

$$||(I - P_Z)\,A|| \leq C(m, n, k)^{1/(2\,q+1)}\,\sigma_{k+1}$$

when $Z = (A^* A)^q A\,\Omega$.

10-log of errors

incurred when using

the power method with:

$q = 0$ in pink

$q = 1$ in blue

$q = 2$ in green

$q = 3$ in black

ℓ

The matrix A being analyzed is a 9025×9025 matrix arising in image processing.
(It is a graph Laplacian on the manifold of 9×9 patches.)
The red crosses mark the singular values of A.

Some observations ...

The observation that a "thin" Gaussian random matrix to high probability is well-conditioned is at the heart of the celebrated Johnson-Lindenstrauss lemma:

Lemma: *Let ε be a real number such that $\varepsilon \in (0, 1)$, let n be a positive integer, and let k be an integer such that*

$$k \geq 4 \left(\frac{\varepsilon^2}{2} - \frac{\varepsilon^3}{3} \right)^{-1} \log(n). \tag{3}$$

Then for any set V of n points in $\mathbb{R}^d$, there is a map $f : \mathbb{R}^d \to \mathbb{R}^k$ such that

$$(1 - \varepsilon) \, ||u - v||^2 \leq ||f(u) - f(v)|| \leq (1 + \varepsilon) \, ||u - v||^2, \qquad \forall \; u, \, v \in V. \tag{4}$$

Further, such a map can be found in randomized polynomial time.

It has been shown that an excellent choice of the map f is the linear map whose coefficient matrix is a $k \times d$ matrix whose entries are i.i.d. Gaussian random variables (see, *e.g.* Dasgupta & Gupta (1999)).
When k satisfies, (3), this map satisfies (4) with probability close to one.

The related Bourgain embedding theorem shows that such statements are not restricted to Euclidean space:

Theorem:. *Every finite metric space (X, d) can be embedded into ℓ^2 with distortion $O(\log n)$ where n is the number of points in the space.*

Again, random projections can be used as the maps.

The Johnson-Lindenstrauss lemma (and to some extent the Bourgain embedding theorem) expresses a theme that is recurring across a number of research areas that have received much attention recently. These include:

- Compressed sensing (Candès, Tau, Romberg, Donoho).
- Approximate nearest neighbor search (Jones, Rokhlin).
- Geometry of point clouds in high dimensions (Coifman, Jones, Lafon, Lee, Maggioni, Nadler, Singer, Warner, Zucker, *etc*).
- Construction of multi-resolution SVDs.
- Clustering algorithms.
- Search algorithms / knowledge extraction.

Note: Omissions! No ordering. Missing references. Etc etc.

Many of these algorithms work "unreasonably well."

The randomized algorithm presented here is close in spirit to randomized algorithms such as:

- Randomized quick-sort.
  (With variations: computing the median / order statistics / *etc.*)
- Routing of data in distributed computing with unknown network topology.
- Rabin-Karp string matching / verifying equality of strings.
- Verifying polynomial identities.

Many of these algorithms are of the type that it is the *running time* that is stochastic. The quality of the final output is excellent.

The randomized algorithm that is perhaps the best known within numerical analysis is Monte Carlo. This is somewhat lamentable given that MC is often a "last resort" type algorithm used when the curse of dimensionality hits — inaccurate results are tolerated simply because there are no alternatives. (These comments apply to the traditional "unreformed" version of MC — for many applications, more accurate versions have been developed.)

Observation: Mathematicians working on these problems often focus on minimizing the distortion factor

$$\frac{1+\varepsilon}{1-\varepsilon}$$

arising in the Johnson-Lindenstrauss bound:

$$(1-\varepsilon)\,||u-v||^2 \leq ||f(u)-f(v)|| \leq (1+\varepsilon)\,||u-v||^2, \qquad \forall\ u,\, v \in V.$$

In our environments, we do not need this constant to be particularly close to 1. It should just not be "large" — say less that 10 or some such.

This greatly reduces the number of random projections needed! Recall that

$$\text{number of samples required } \sim \frac{1}{\varepsilon^2}\,\log(N).$$

Observation: Multiplication by a random unitary matrix reduces any matrix to its "general" form. All information about the singular vectors vanish. (The singular *values* remain the same.)

This opens up the possibility for general pre-conditioners —
counterexamples to various algorithms can be disregarded.

The feasibility has been demonstrated for the case of least squares solvers for very large, very over determined systems. (Work by Rokhlin & Tygert, Sarlós, ...)

Work on $O(N^2 \, (\log N)^2)$ solvers of general linear systems is under way.
(Random pre-conditioning + iterative solver.)

May $O(N^2 \, (\log N)^2)$ matrix inversion schemes for general matrices be possible?

Observation: Robustness with respect to the quality of the random numbers.

The assumption that the entries of the random matrix are i.i.d. normalized Gaussians simplifies the analysis since this distribution is invariant under unitary maps.

In practice, however, one can use a low quality random number generator. The entries can be uniformly distributed on $[-1, 1]$, they be drawn from certain Bernouilli-type distributions, *etc.*

Remarkably, they can even have enough internal structure to allow fast methods for matrix-vector multiplications. For instance:

- Subsampled discrete Fourier transform.
- Subsampled Walsh-Hadamard transform.
- Givens rotations by random angles acting on random indices.

This was exploited in "Algorithm 2" (and related work by Ailon and Chazelle).
Our theoretical understanding of such problems is unsatisfactory.
Numerical experiments perform *far* better than existing theory indicates.

Even though it is thorny to *prove* some of these results (they draw on techniques from numerical analysis, probability theory, functional analysis, theory of randomized algorithms, *etc*), work on randomized methods in linear algebra is progressing fast.

Important: Computational prototyping of these methods is extremely simple.

- Simple to code an algorithm.
- They work so well that you immediately know when you get it right.

Current research directions:

- Acceleration of BLAS / LINPACK functions.
  - May actually soon be integrated in Matlab and Mathematica.
- Construction of reduced models for physical phenomena ("model reduction").
  - Wave propagation through media with periodic micro-structures.
  - Scattering problems involving multiple scatterers.
- New estimates on spectral properties of random matrices.

Potential "killer" application:

Approximation of very large very noisy data sets stored on disk or streamed.

The randomized algorithms solve two fundamental limitations of existing methods:

- Propagation of rounding errors.
- Data flow — we can only see the data once!

Important applications that cannot be solved with existing technology:
Image and video processing / network analysis / statistical data processing / ...