Fast Matrix Computations via Randomized Sampling

Gunnar Martinsson, The University of Colorado at Boulder

Computational science — background

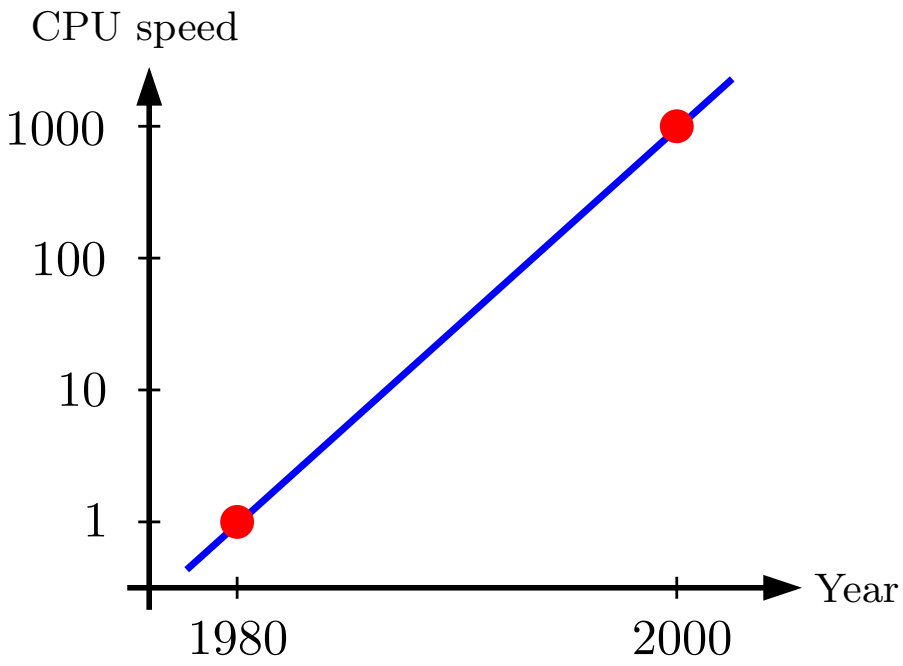
One of the principal developments in science and engineering over the last couple of decades has been the emergence of computational simulations.

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

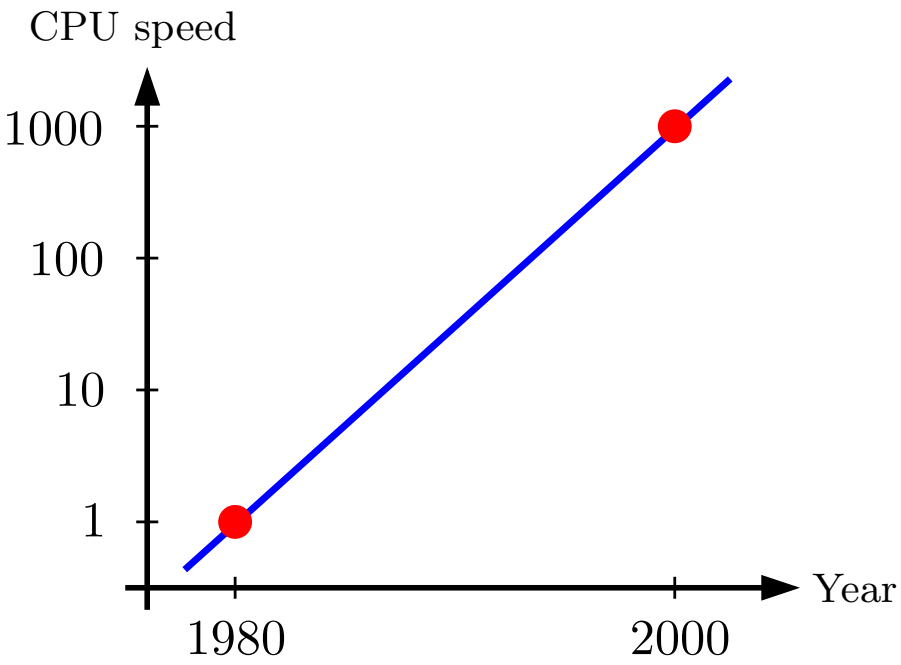
This shift towards computational modelling has in many areas led to dramatic cost savings and improvements in performance.

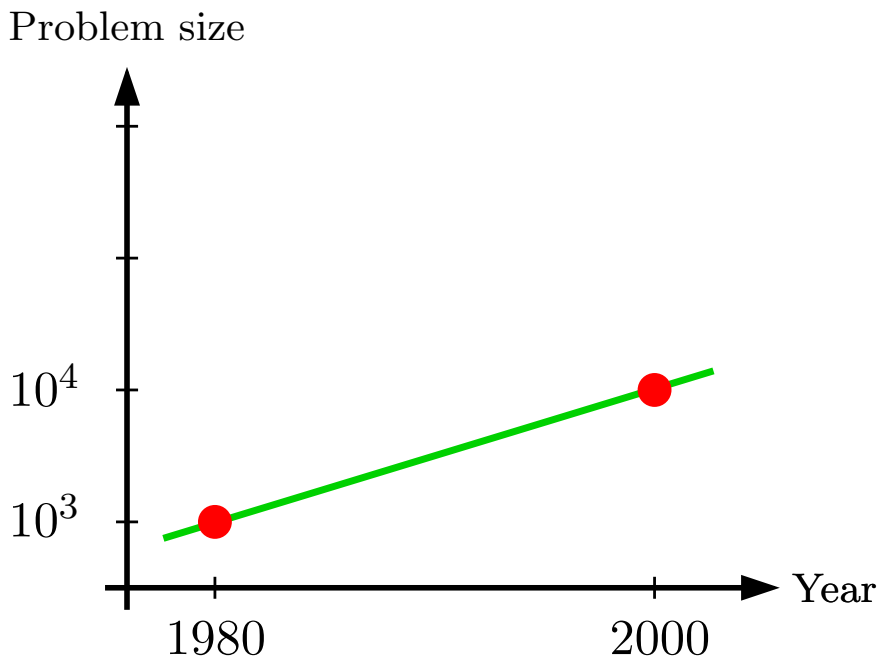
What enabled all this was the development of faster more powerful computers, *and the development of faster algorithms*.

Growth of computing power and the importance of algorithms



Growth of computing power and the importance of algorithms



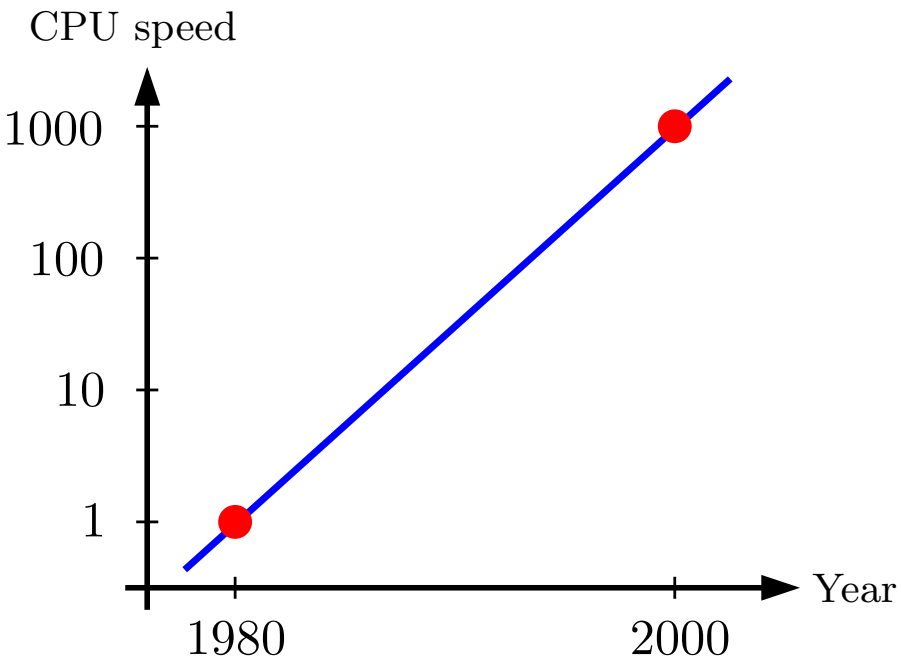


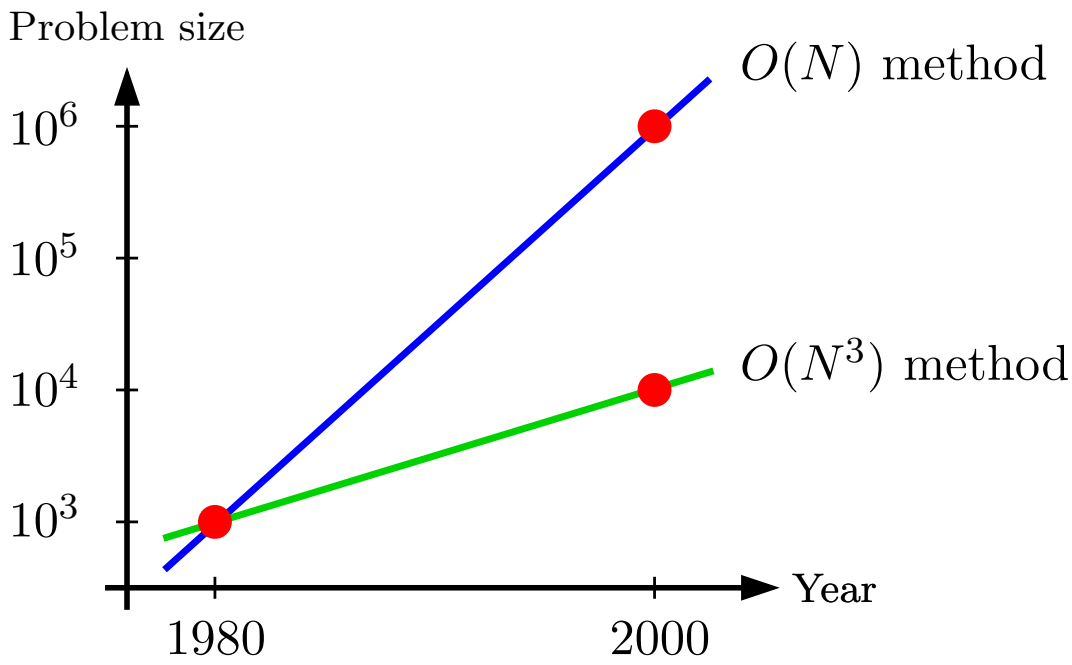
Consider the computational task of solving a linear system of N algebraic equations with N unknowns.

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

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

Growth of computing power and the importance of algorithms





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

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

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

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

Definition of the term "fast":

We say that a numerical method is "fast" if its computational speed scales as $O(N)$, or possibly $O(N\log(N))$ or $O(N(\log N)^2)$, as the problem size N grows.

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 described in this talk parallelize naturally. In fact, this is a principal selling point!

Topic of this talk: "Fast" algorithms for standard matrix operations such as:

- Matrix-vector multiplication.
- Matrix-inversion.
- Matrix factorization.
  - LU
  - QR
  - *etc*
- Spectral decompositions.
  - Eigen-decomposition: $A = V\,D\,V^{-1}$.
  - Singular Value Decomposition: $A = U\,D\,V^{\mathrm{t}}$.

Topic of this talk: "Fast" algorithms for standard matrix operations such as:

- Matrix-vector multiplication. $O(N^2)$
- Matrix-inversion. $O(N^3)$
- Matrix factorization. $O(N^3)$
  - LU
  - QR
  - *etc*
- Spectral decompositions. $O(N^3)$
  - Eigen-decomposition: $A = V\,D\,V^{-1}$.
  - Singular Value Decomposition: $A = U\,D\,V^{\mathrm{t}}$.

The computational cost of performing each task using well-established techniques on a general $N \times N$ matrix A are given in red above.

Not the topic of this talk:

There exist methods for multiplying two general $N \times N$ matrices in less than $O(N^3)$ operations:

- **Strassen:** $O(N^{2.807})$
- **Coppersmith-Winograd:** $O(N^{2.376})$

These methods can be adapted to perform accelerated matrix inversions, factorizations, *etc.*

We are interested in *much* faster methods — $O(N)$, $O(N \log N)$, $O(N \log^2 N)$.

This of course requires some kind of "structure" in the matrix (less than one might have thought).

One of the simplest forms of "structure" in a matrix is rank-deficiency:

Suppose that the $N \times N$ matrix A has rank k, meaning that it allows the factorization

$$\begin{array}{ccccc} A & = & B & & C. \\ N \times N & & N \times k & & k \times N \end{array}$$

Then, *once you have constructed the factors B and C*,

- multiplying A and a vector costs $O(N\,k)$,
- computing the LU-factorization of A costs $O(N\,k^2)$,
- computing the singular value decomposition of A costs $O(N\,k^2)$.

The catch: Standard methods for computing such factors cost $O(N^2\,k)$.

Question: Can B and C be constructed faster?

Caveat:

Most matrices that we will discuss will not have rank exactly equal to k.

Instead of the factorization

$$\underset{N \times N}{A} = \underset{N \times k}{B} \; \underset{k \times N}{C}, \tag{1}$$

most matrices of interest to us will satisfy something along the lines of

$$\underset{N \times N}{A} = \underset{N \times k}{B} \; \underset{k \times N}{C} + \underset{N \times N}{R}, \tag{2}$$

where the "remainder" matrix R is smaller than some preset precision ε:

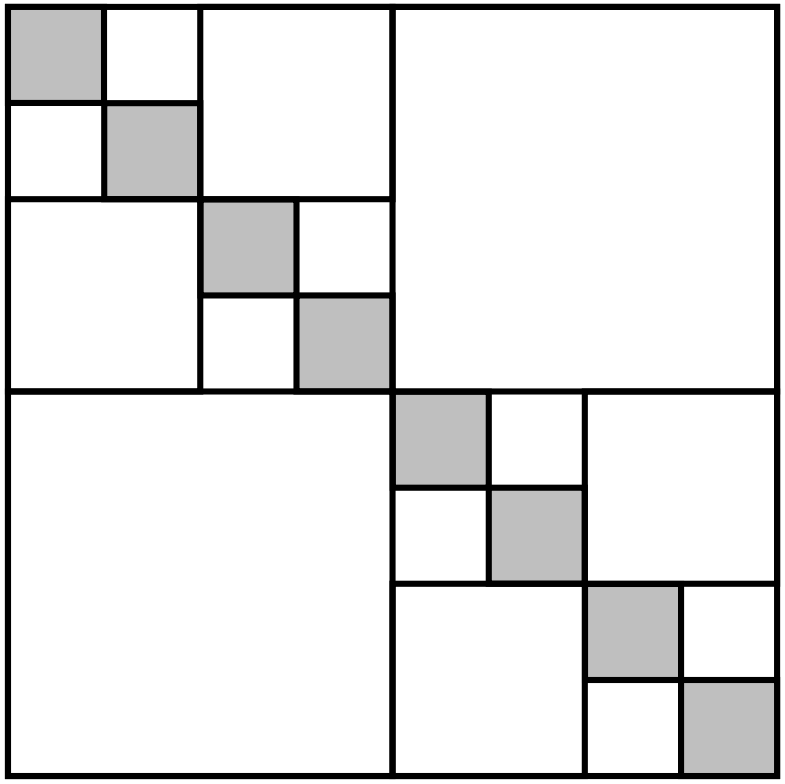
$$||R|| \leq \varepsilon. \tag{3}$$

In our examples, ε is typically very small, say $\varepsilon \sim 10^{-10}$.

If A satisfies (2) for some R that satisfies (3), we say that "A has ε-rank k".

In many applications, the matrix A that we are interested in does not itself have low rank — *but it can be tesselated into blocks that do*.

The following figure shows one of the most common tessellations:



For the purposes of this talk, we call a matrix with this property "block rank-deficient".

. . . shiv's picture . . .

"Block rank-deficient" matrices are ubiquitous in scientific computing:

- Numerical methods for elliptic and parabolic PDEs.
- Models of N particles with linear interactions.
- Signal processing (with matrices representing transforms and filters).
- Matrices that model networks are typically *sparse* — factorizing them often leads to dense but "block rank-deficient" matrices.

The structure of "block rank-deficient" can be exploited to construct accelerated — $O(N)$ or $O(N\,(\log N)^p)$ — algorithms for standard operations:

- Matrix-vector multiply. ("Fast Multipole Method", "Barnes-Hut", *etc.*)
- [New!] Matrix inversion.
- [New!] Matrix-matrix multiplication.
- [New!] Matrix factorization (QR, LU, Cholesky, *etc*).
- [Hopefully upcoming...] Partial eigenvalue decompositions, SVDs.

Example 1 — Matrices approximating elliptic differential operators:

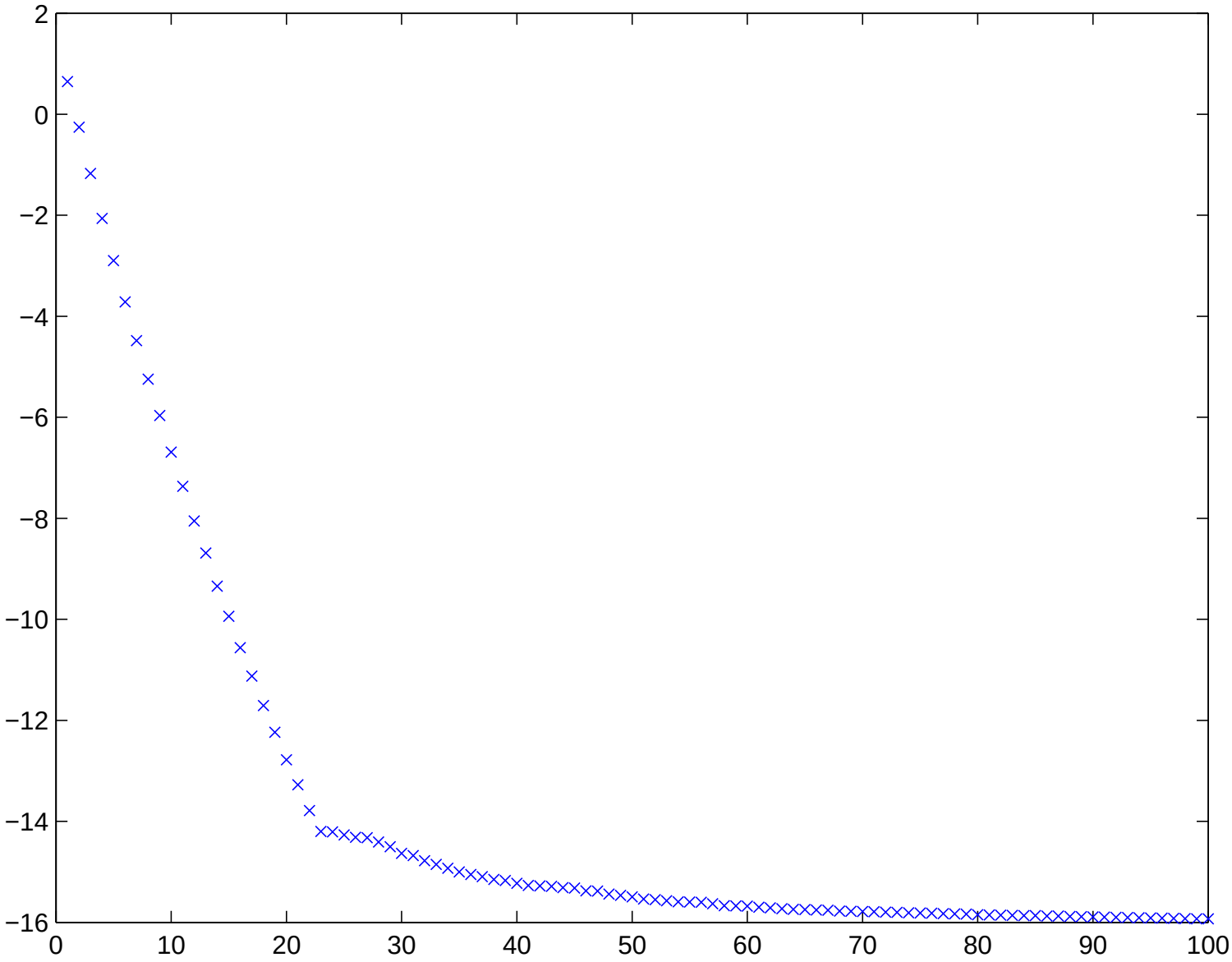
Let L be the standard five-point stencil (discrete Laplacian) on a 50×50 grid:

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

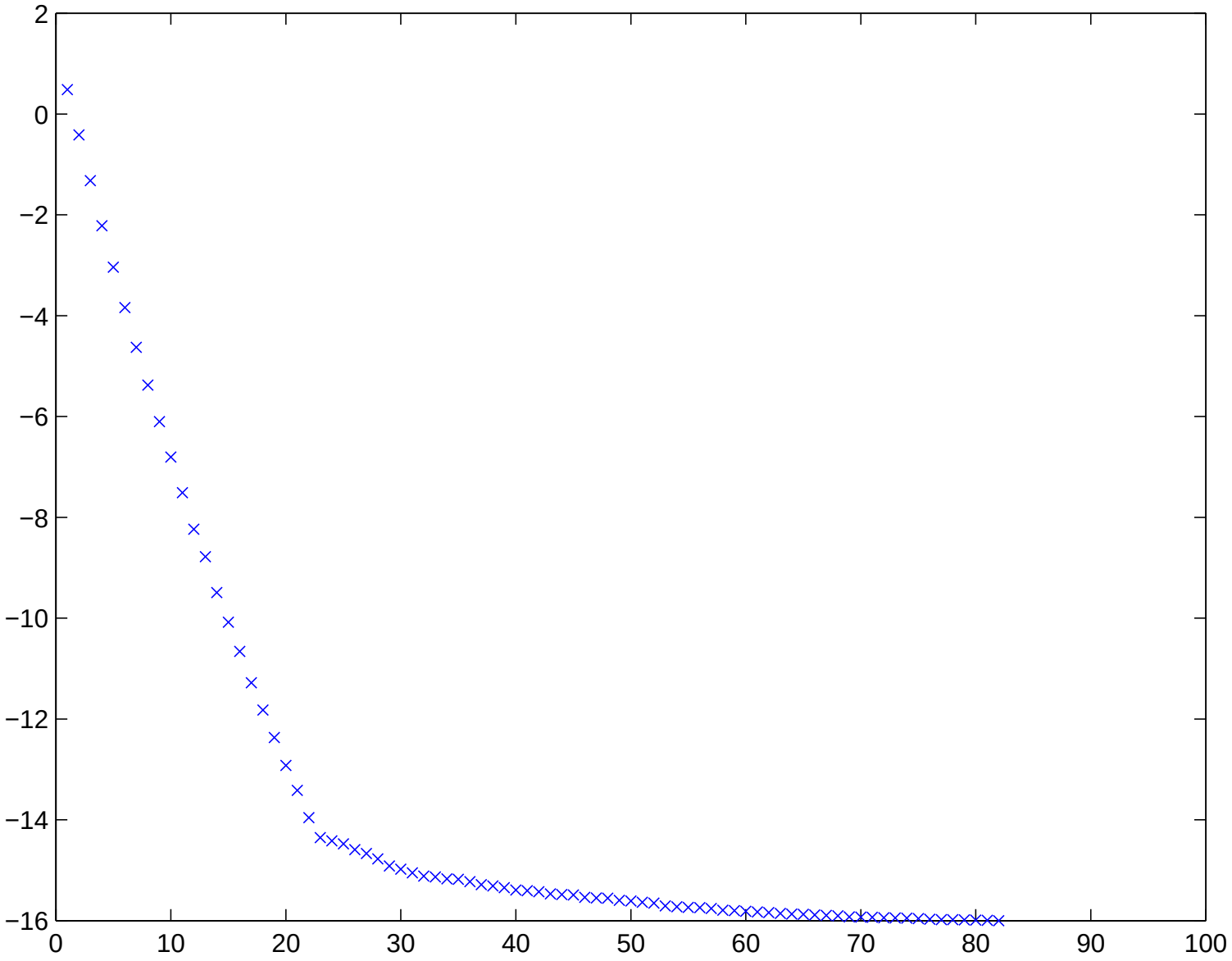
Let A be the inverse of L, and partition it:

$$A = L^{-1} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}.$$

We consider the 625×625 submatrix A_{14} of the $2\,500 \times 2\,500$ matrix A.



The 10-logarithm of the singular values of A_{14}.



The 10-logarithm of the singular values of A_{14} — now with *random coefficients*.

A fast direct solver for finite element matrices
(Martinsson, 2007)

We have constructed an $O(N\,(\log N)^2)$ scheme for directly inverting the matrix L.

For $N = 1\,000\,000$, the scheme requires 4 minutes for the initial inversion (on an 2.8 GHz Pentium IV desktop with 512 Mb of RAM) performed at single precision (seven correct digits).

Subsequent problems that are loaded on the boundary only can be solved in 0.03 seconds (provided that only the solution on the boundary is sought).

The scheme is very primitive. Work that will improve the asymptotic time requirement to $O(N\,\log N)$ (or possibly $O(N)$) is under way. The constants will be much smaller.

Works for a wide range of elliptic equations.

Similar results have very recently been obtained by Gu, Chandrasekaran, Xia, Li.

Example 2 — Integral equation methods for elliptic PDEs:

Integral operators (single layers, double layers, *etc*) associated with PDEs such as

- Laplace's equation,
- Helmholtz equation (at low and intermediate frequencies),
- the equations of elasticity,
- the Yukawa equation,
- *etc*

turn into "block rank-deficient" matrices upon discretization.

Discretizations of Green's functions on bounded domains ...

Approximations of Dirichlet-to-Neumann operators ...

Etc, etc, ...

Block rank-deficient matrices abound in this environment.

A fast direct solver for boundary integral operators
(Martinsson & Rokhlin, 2004)

Computational results for the double layer potential associated with an exterior Laplace problem on a smooth contour:

Relative accuracy: 10^{-10}

Problem size: $102\,400 \times 102\,400$.

Time for initial inversion: 19 seconds.

Time for applying the inverse: 0.17 seconds.

Computational results for the combined field equation associated with an exterior Helmholtz problem on a smooth contour 200 wave-lengths in size:

Relative accuracy: 10^{-8}

Problem size: $25\,600 \times 25\,600$.

Time for initial inversion: 7 minutes.

Time for applying the inverse: 0.79 seconds.

Note: This method is not "fast" at very large wave numbers.

Example 3 — Toeplitz matrices:

(Martinsson, Rokhlin, Tygert, 2005)

Let $\ldots, a_{-2}, a_{-1}, a_0, a_1, a_2, \ldots$ be complex numbers.

For a positive integer N, let A denote the $N \times N$ matrix with entries

$$A_{ij} = a_{j-i}.$$

For instance, for $N = 4$, we get

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_{-1} & a_0 & a_1 & a_2 \\ a_{-2} & a_{-1} & a_0 & a_1 \\ a_{-3} & a_{-2} & a_{-1} & a_0 \end{bmatrix}.$$

Then the Fourier transform of A,

$$\hat{A} = F_N \, A \, F_N^*$$

is a structured matrix. (A itself is not necessarily "structured".)

Example 4 — Sampling of "sinc" matrix:

Let $\{x_j\}_{j=1}^N$ denote real numbers such that $-\infty < x_1 < x_2 < \cdots < x_{N-1} < x_N < \infty$.

Define for a real positive number c the $N \times N$ matrix A by

$$A_{ij} = \begin{cases} c & i = j, \\ \dfrac{\sin\big(c\,(x_i - x_j)\big)}{x_i - x_j}, & i \neq j. \end{cases}$$

Then A is a structured matrix.

Prolate spheroidals are eigenfunctions of the sinc operator — connection to band-filtering signals / image processing / customized Fourier transforms / *etc.* *(Work in progress.)*

Recall:

- Matrix-vector multiplies. (As in FMM, Barnes-Hut, panel clustering, *etc.*)
- [New!] Matrix-matrix multiplies.
- [New!] Matrix factorizations (QR, LU, Cholesky, *etc*).
- [New!] Matrix inversions.

Fast matrix-vector multiplication schemes such as the Fast Multipole Method and Barnes-Hut exploit known analytic properties of the off-diagonal blocks (they are oftentimes samples of a known kernel function).

A key difficulty in constructing matrix inversion schemes is that such precise knowledge about the kernel of the inverse functions is typically not available.

How then, do you construct bases for the off-diagonal blocks?

In the remainder of this talk, we will consider only the problem where the entire matrix itself is rank-deficient to some precision ε.

Unsupported assertion: The techniques we are about to present can in many important environments be adapted to "block rank-deficient" matrices — the only thing you need is a fast global matrix-vector multiplier. (Martinsson 2008)

The specific technical problem that we will address is the following:

Suppose that an $N \times N$ matrix A has ε-rank k where $k \ll N$.

We seek to determine factors B and C such that

$$\begin{array}{ccccc} A & \approx & B & C. \\ N \times N & & N \times k & k \times N \end{array}$$

We will discuss two different environments:

Case 1:

We have a technique for rapidly performing matrix-vector products:

$$x \mapsto A\,x.$$

If the cost of a matrix-vector product is T_{mult}, then the algorithm for this case has complexity $O(T_{\text{mult}}\,k)$.

Case 2:

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

The complexity of this algorithm is $O(N^2 \log(k))$.

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

There is a well-established method for this problem: Lanczos.

This algorithm is sometimes $O(T_{\text{mult}}\,k)$, but the method that we will describe is more robust, faster, and is naturally parallelizable.

The method that we propose is based on randomized sampling.

This means that it has a non-zero probability of giving an incorrect answer.

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^{-10} are standard.

Definition: We say that a vector $\omega \in \mathbb{R}^N$ is a random direction vector if

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_N \end{bmatrix},$$

where the numbers $\{\omega_j\}_{j=1}^N$ are random variables drawn independently from a normalized Gaussian distribution.

Note that the probability distribution of ω is isotropic in the sense that it is invariant under rotations in $\mathbb{R}^N$.

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

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

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

Form the length-m vectors

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

Each y_j is a "random linear combination" of columns of A.

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

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

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

What is remarkable is how fast this probability approaches one.

How to measure "how well we are doing":

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

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

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

The "error" after l steps is then

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

The quantity e_l should be compared to the minimal error

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

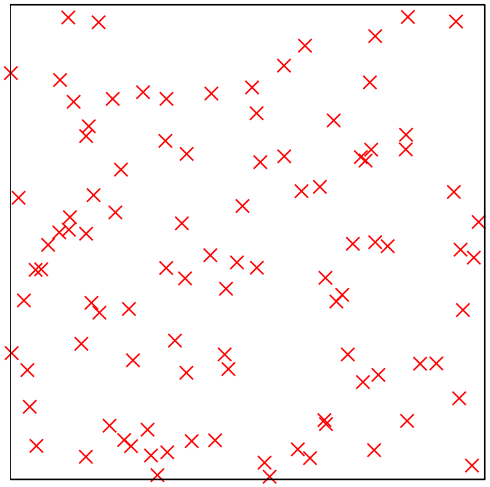
In reality, computing e_l is not affordable. Instead, we compute something like

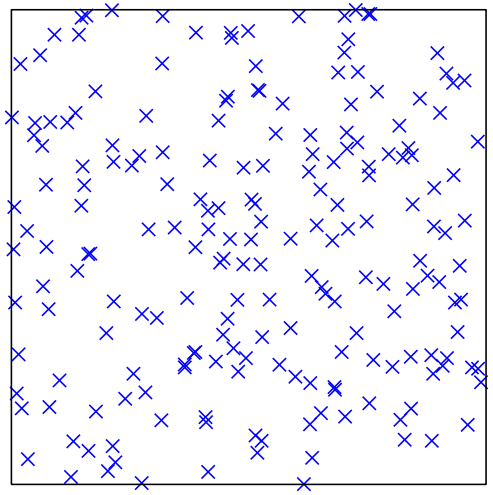
$$f_l = \max_{1\leq j\leq 10} \left|\left|\left(I - V_l\,V_l^{\mathrm{t}}\right) y_{l+j}\right|\right|.$$

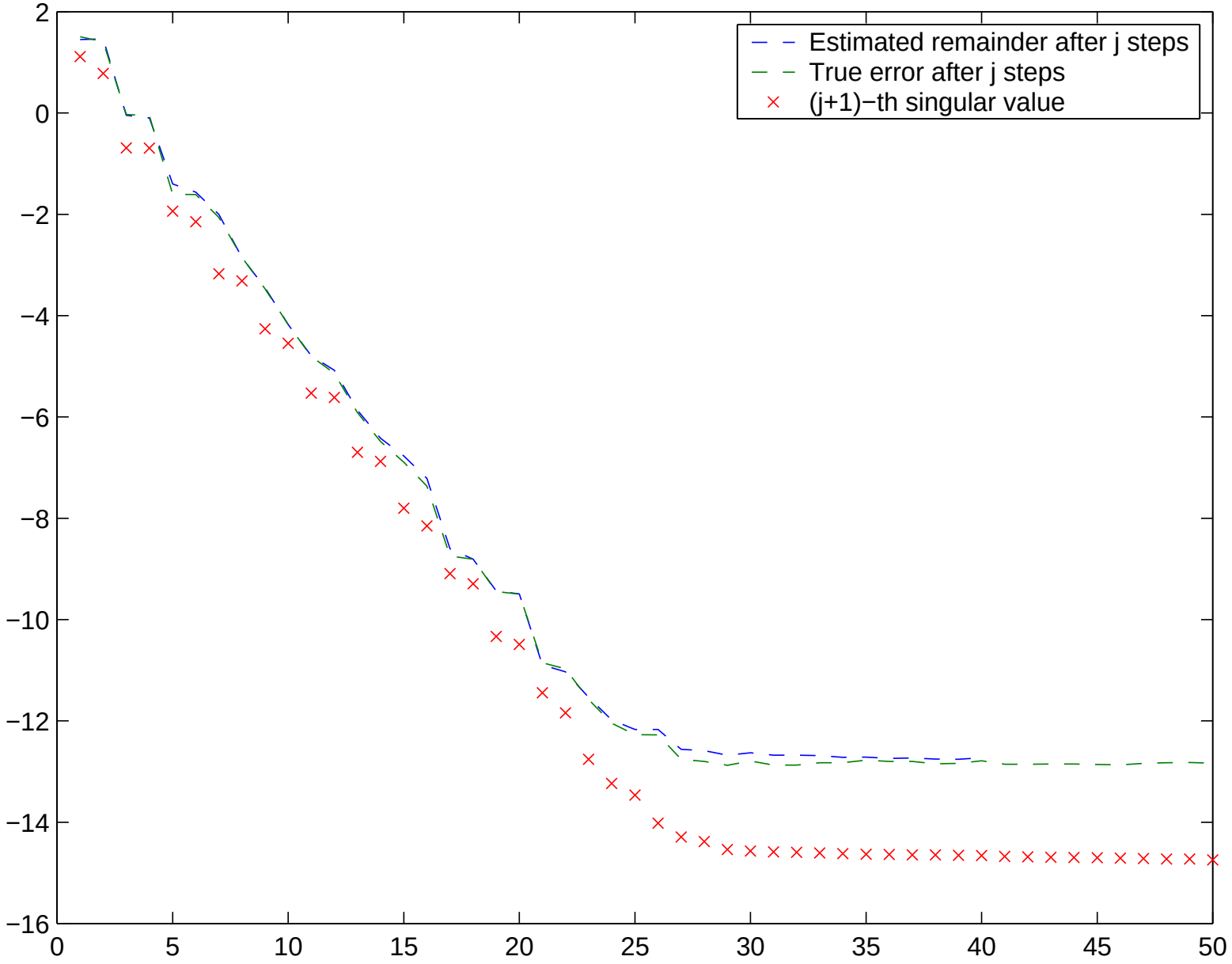
The computation stops when we come to an l such that $f_l < \varepsilon/\sqrt{N}$.

To illustrate the performance, we study the following particular choice of A:

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







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

Was this just a lucky realization?

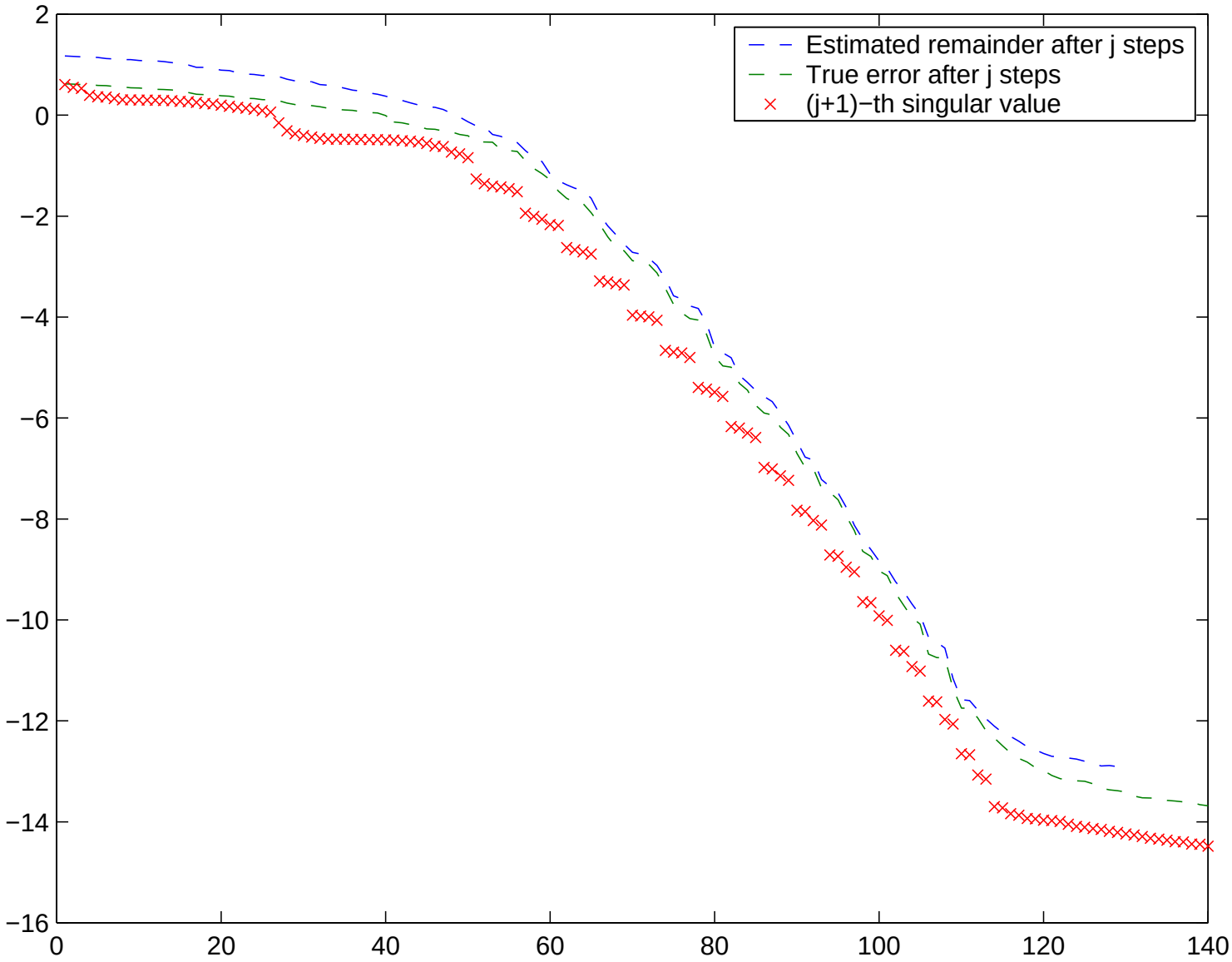
We collected statistics from 1 000 000 realizations:

(Recall that the exact ε-rank is 34.)

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

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

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



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

But what about situations where you do not have a fast matrix-vector multiplier?

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.

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

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

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

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

$$\underset{N \times l}{Y} = \underset{N \times N}{A} \quad \underset{N \times l}{\Omega}.$$

Key points:

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

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 $N \log(l)$ operations.

What is this "random but structured" matrix Ω?

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

where,

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

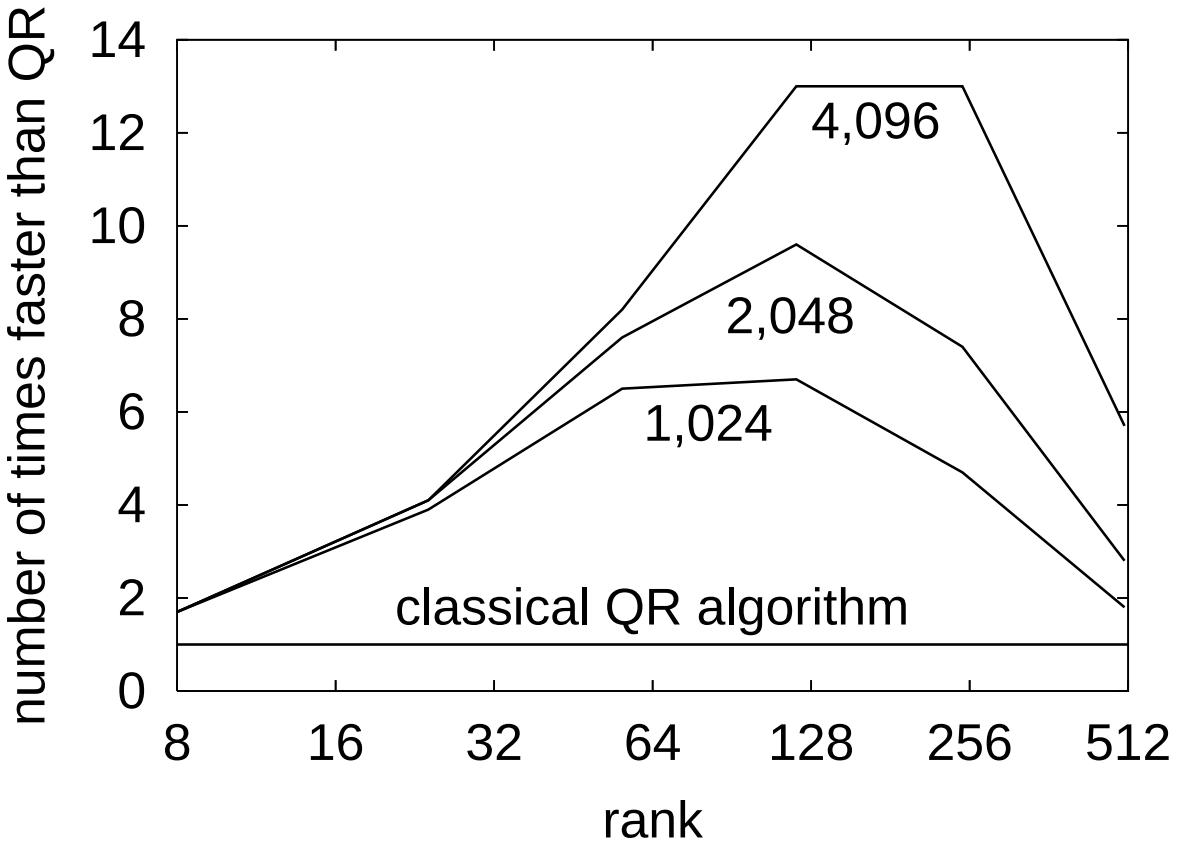
- F is the discrete Fourier transform, $F_{jk} = e^{-2\pi i(j-1)(k-1)/N}$.

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

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

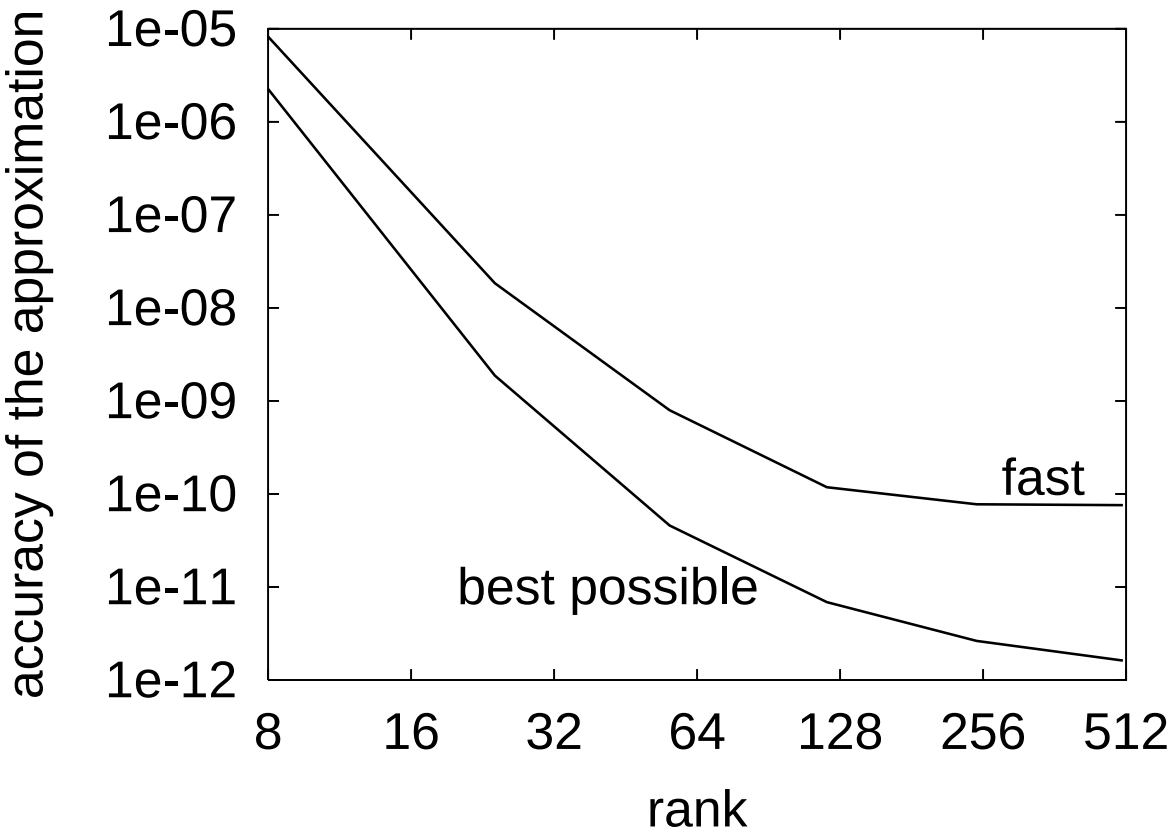
This idea was described by Nir Ailon and Bernard Chazelle (2006).
There is also related recent work by Sarlós (on randomized regression).

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.

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 $N \times N$ matrix.

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

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

Let W be an $N \times l$ 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

$$||A - W\,W^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{l\,N}\;\sigma_{k+1},$$

with probability at least

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

where φ is a decreasing function satisfying

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

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

In substantiating the theorem, the singular value decomposition of A is useful.

Recall that any $N \times N$ matrix A allows a factorization

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

$U = [u_1\ u_2\ \cdots\ u_N]$ is a unitary matrix whose columns are the "left singular vectors".

$V = [v_1\ v_2\ \cdots\ v_N]$ is a unitary matrix whose columns are the "left singular vectors".

D is a diagonal matrix whose diagonal entries σ_1, σ_2, $\ldots$, σ_N are the "singular values".

The singular values are ordered so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N \geq 0$.

Then

$$A = [u_1\ u_2\ \cdots\ u_N] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_N \end{bmatrix} \begin{bmatrix} v_1^{\mathrm{t}} \\ v_2^{\mathrm{t}} \\ \vdots \\ v_N^{\mathrm{t}} \end{bmatrix} = \sum_{j=1}^{N} \sigma_j\, u_j\, v_j^{\mathrm{t}}.$$

A	$=$	U	D	V^{t}
		unitary	*diagonal*	*unitary*

The claim of the theorem is that

$$||A - W\,W^{\mathrm{t}}\,A||_2 \leq 10\,\sqrt{l\,N}\,\sigma_{k+1}.$$

The norm is invariant under rotations:

$$||A - W\,W^{\mathrm{t}}\,A||_2 = ||U^{\mathrm{t}}\,A - U^{\mathrm{t}}\,W\,W^{\mathrm{t}}\,A||_2 = ||U^{\mathrm{t}}\,A - (U^{\mathrm{t}}\,W)\,(U^{\mathrm{t}}\,W)^{\mathrm{t}}\,(U^{\mathrm{t}}\,A)||_2$$

Recall that W was defined as an ON-basis for $A\,\Omega$.
It follows that $U^{\mathrm{t}}\,W$ is an ON-basis for $U^{\mathrm{t}}\,A\,\Omega$.

The upshot is that we can henceforth assume that $A = D\,V^{\mathrm{t}}$.

From the previous slide: We can assume that $A = D\,V^{\mathrm{t}}$.

Then $A\,\Omega = D\,V^{\mathrm{t}}\,\Omega$.

Now note that the entries of $V^{\mathrm{t}}\,\Omega$ are themselves i.i.d. normalized Gaussian random variables (since the probability distribution is directionally uniform).

As a consequence, we can replace Ω by $V^{\mathrm{t}}\,\Omega$ throughout the proof.
The effect is the same as simply assuming that $A = D$.

We will without loss of generality assume that A is a diagonal matrix.

Assuming that A is a diagonal matrix, we find that

$$A\,\Omega = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_N \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \cdots & \omega_{1l} \\ \omega_{21} & \omega_{22} & \cdots & \omega_{2l} \\ \vdots & \vdots & & \vdots \\ \omega_{N1} & \omega_{N2} & \cdots & \omega_{Nl} \end{bmatrix}.$$

The question becomes: How large must l be before we can say with high probability that the columns of an $N \times l$ matrix populated with i.i.d. Gaussian random variables with high probability sample the first k coordinates?

At this point, there is a rich literature in probability theory to draw on. The arguments are based on "concentration of mass" and the key insight is that "thin" random matrix tend to be well-conditioned (or at least have well-conditioned sub-matrices) and sample high-dimensional space with almost optimal efficiency.

The observation that a "thin" matrix populated with i.i.d. Gaussian random variables is to high probability 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*

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

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

$$(1 - \varepsilon) \, ||u - v||^2 \leq ||f(u) - f(v)|| \leq (1 + \varepsilon) \, ||u - v||^2, \qquad \forall \; u, v \in V. \tag{5}$$

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, (4), this map satisfies (5) 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).
- Geometry of point clouds in high dimensions (Coifman, Maggioni, Jones, *etc*).
- Construction of multi-resolution SVDs.
- Clustering algorithms.
- Search algorithms / knowledge extraction.
- Approximate nearest neighbor search.

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 among numerical analysis is Monte Carlo. This is slightly ironic given that MC is often a "last resort" type algorithm when the curse of dimensionality hits — we accept inaccurate results simply because we have no options.
(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 tend to be very focussed 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.

We are currently working on a number of applications:

- Construction of fast algorithms for solving PDEs numerically.
- Acceleration of classical linear algebra routines.
- Construction of reduced models for physical phenomena ("model reduction").
  - Modeling of percolating micro-structures.
  - Wave propagation through media with periodic micro-structures.
  - Crack propagation through composite materials.
  - Scattering problems involving multiple scatterers.
- Analysis of statistical data sets — "principal component analysis" — finding "latent" variables.
- Analysis of network matrices (such as ones representing the link structure of the World Wide Web) — knowledge extraction

Summary

Randomized algorithms for approximating rank-deficient matrices.
Work for "block rank-deficient" matrices as well, and thus enable the development of $O(N)$ methods for a wide range of matrix computations.

The methods presented are closely related to recent developments in functional analysis and random matrix theory. The methods described are in fact much simpler than other recent work. However, they appear to have been overlooked.

The randomized methods have resolved a number of long-standing problems in numerical analysis, for instance:

- Definitively overcome issues relating to stability and parallelization of Lanzcos.
- Fast compression and inversion of "block rank-deficient" matrices.
- Reduction of CPU time requirement of partial spectral decompositions, QR factorizations, *etc*, from $O(N^2\,k)$ to $O(N^2\log(k))$. (Rokhlin, Tygert, Liberty)

I believe there is much more to come.