Fast Matrix Computations via Randomized Sampling

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)

Franco Woolfe (Goldman Sachs)

Support: IPAM (via "Mathematics of Knowledge and Search Engines" program)
NSF-DMS #0748488 and #0610097

Topic of the talk:
Development of new fast methods for performing the linear algebraic operations arising in algorithms for extracting information from large data sets.

Applications of interest:

- Clustering of data points.
- Nearest neighbor search.
- Finding "local coordinates" given noisy sample points from a manifold.
- Searches for items in a set that are "similar" in some specified sense.

The common theme is that we are in some sense studying the geometry of clouds of points in high dimensional space.

Some problems of this type can be cast as matrix computations — determination of eigenvectors or singular vectors, matrix inversions, *etc.*
(Some of them, not all!)

Example: "Page Rank" and similar algorithms

The "page rank" algorithm is a well known technique for analyzing the link structure of the World Wide Web:

- Enumerate "all" web pages, $i = 1, 2, \ldots, N$.
- Let ℓ_j denote the number of links from page j.
- Form the "adjacency matrix" L by setting

$$\mathsf{L}_{ij} = \begin{cases} 1/\ell_j, & \text{if web page } j \text{ links to web page } i, \\ 0 & \text{otherwise.} \end{cases}$$

- Fix a damping factor d. (A common choice is $d = 0.85$.)
- The vector of "page ranks" p is now the solution of the eigenequation

$$p = \left(\frac{1-d}{N} \, \mathsf{E} + d \, \mathsf{L} \right) p,$$

where E is the $N \times N$ matrix whose entries are all ones.

There are many variations and improvements to the basic "Page Rank" algorithm.

Whether this is a good way of analyzing the web is a subject of some contention. Many people favor methods not based on linear algebra at all — some argue that many singular vectors must be computed . . .

We do not hitch our wagon to any particular side in this argument, but simply observe that being able to compute a few approximate eigenvectors for Markov matrices of various kinds can be useful.

Example: Population genetics

From: P. Paschou, E. Ziv, E. Burchard, M.W. Mahoney, and P. Drineas

Matrix A recording "Single Nucleotide Polymorphisms" (SNPs).
We study a sequence of base pairs that take on the values, say, "R" and "S".
Matrix entry A_{ij} records the value for person j of an allele pair i:

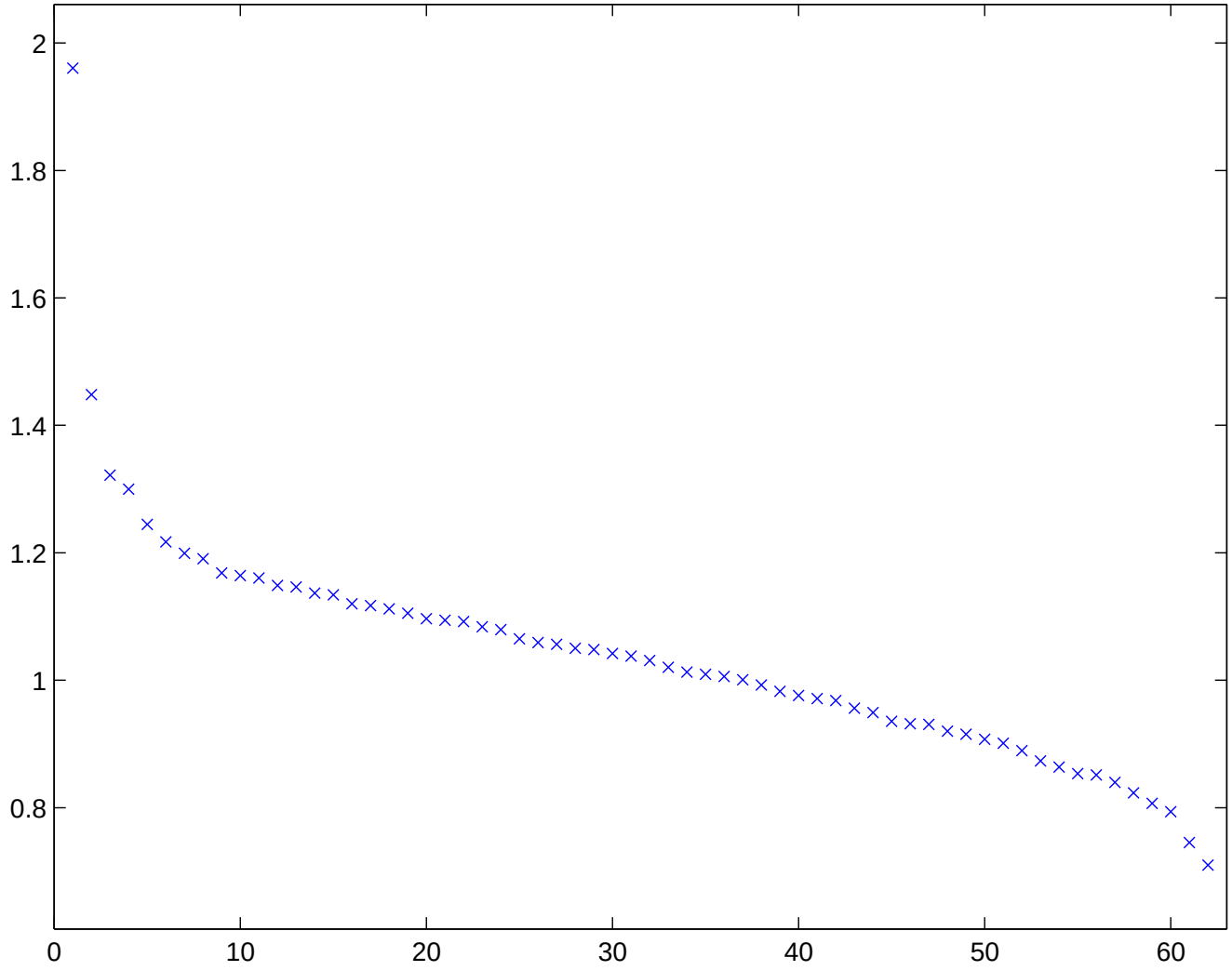
$$\begin{aligned} \text{genotype SS} &\sim A_{ij} = -1 \\ \text{genotype RS} &\sim A_{ij} = 0 \\ \text{genotype RR} &\sim A_{ij} = 1 \end{aligned}$$

In other words

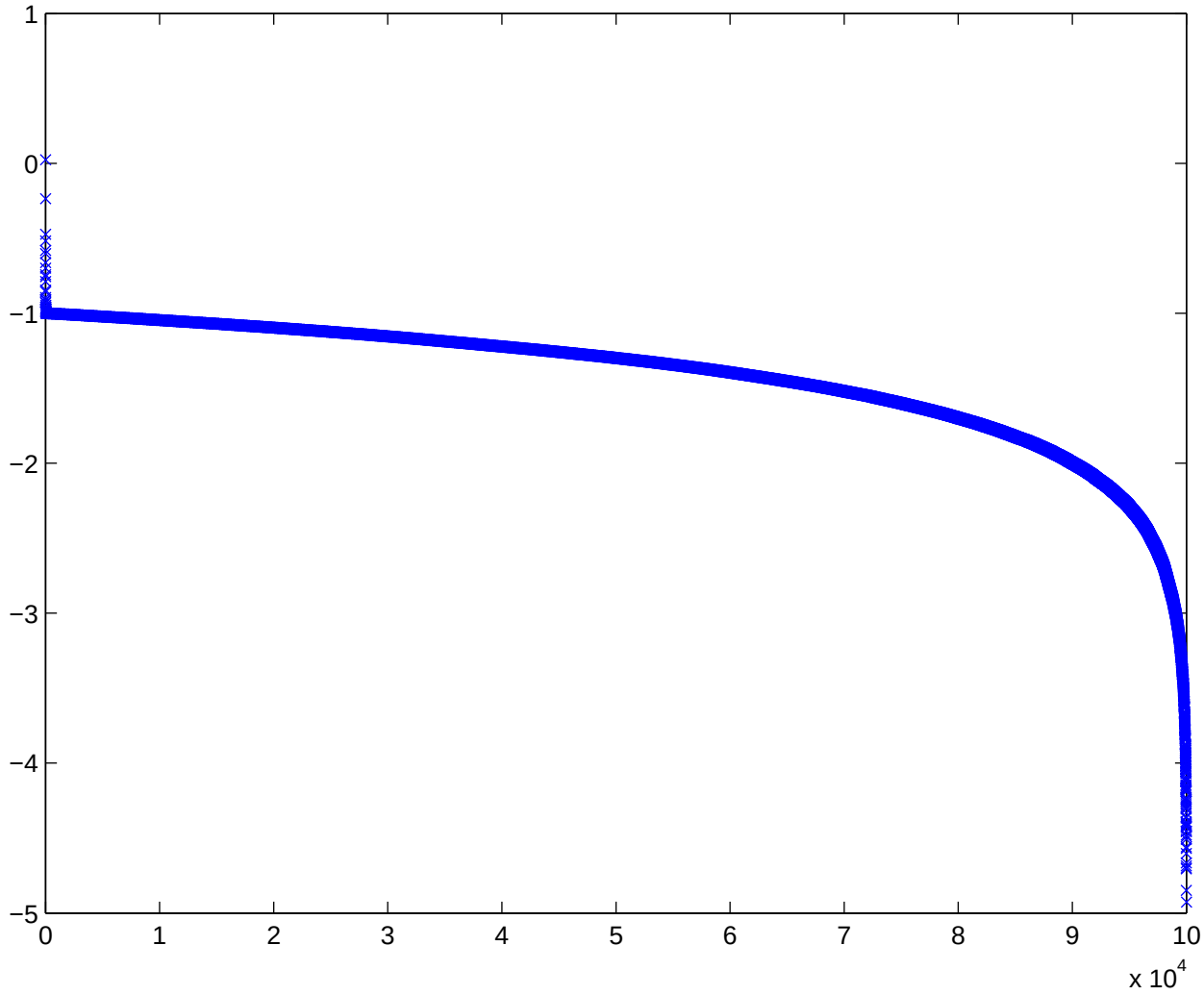
	Person 1	Person 2	Person 3	$\cdots$
Allele 1	RS	RR	RS	$\cdots$
Allele 2	SS	SS	SS	$\cdots$
Allele 3	RS	RS	SS	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

$$\sim \qquad A = \begin{bmatrix} 0 & 1 & 0 & \cdots \\ -1 & -1 & -1 & \cdots \\ 0 & 0 & -1 & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

10-logarithm of singular values of a 400×62 matrix A (400 alleles, 62 people):



Clustering information can be extracted from the first few singular vectors.


1
0
−1
−2
−3
−4
−5
0
1
2
3
4
5
6
7
8
9
10
x 10^4

Similar problems arise in statistical analysis of many different types of data, *e.g.*:

- Genomics.
- Financial data.
- Image analysis.
- ...

The term "Principal Component Analysis" (PCA) is frequently used in this context.

Example: "Diffusion geometry"

A prototypical problem in data analysis is the following:

- A set of points in high dimensional space is given.
- The points are believed to lie (approximately) on a low-dimensional manifold. This manifold is typically not a linear subspace.
- The task is to identify the manifold, for instance by finding a set of "local coordinates".

In the "diffusion geometry" approach (Coifman, Maggioni, Jones, ...), one solves this problem by determining eigenvectors of a "graph Laplacian" acting on the data set.

The linear algebraic problem here is to identify many eigenvectors of a very large sparse matrix with a lot of "noise".

What do we learn from these and similar examples?

Matrix computations arise in a range of different environments.

These applications are challenging to the "traditional" techniques of numerical linear algebra for a number of reasons, including:

- Very large problem sizes.
- Low signal-to-noise ratios.
- Optimal ordering of the points is not known in advance.
  This makes a number of "fast" methods developed for solving the large systems arising from the discretization of PDEs inapplicable without major modifications.

Good news: For the class of problems where the task to be solved is the task of approximating a matrix by a low-rank factorization, techniques whose performance is close to the theoretically optimal one have recently been developed.

The "low-rank approximation problem" arises in a variety of contexts:

- Computing principal components.
- Finding a close to optimal set of spanning rows / spanning columns.
- Determining approximate QR / LU / CUR / LUV / ... factorizations.

These techniques are based on randomized sampling, and we will describe them in some detail.

We will then discuss possible extensions of these techniques to more complex situations, such as the computation of partial spectral decompositions of graph Laplacians and similar matrices.

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

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

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:

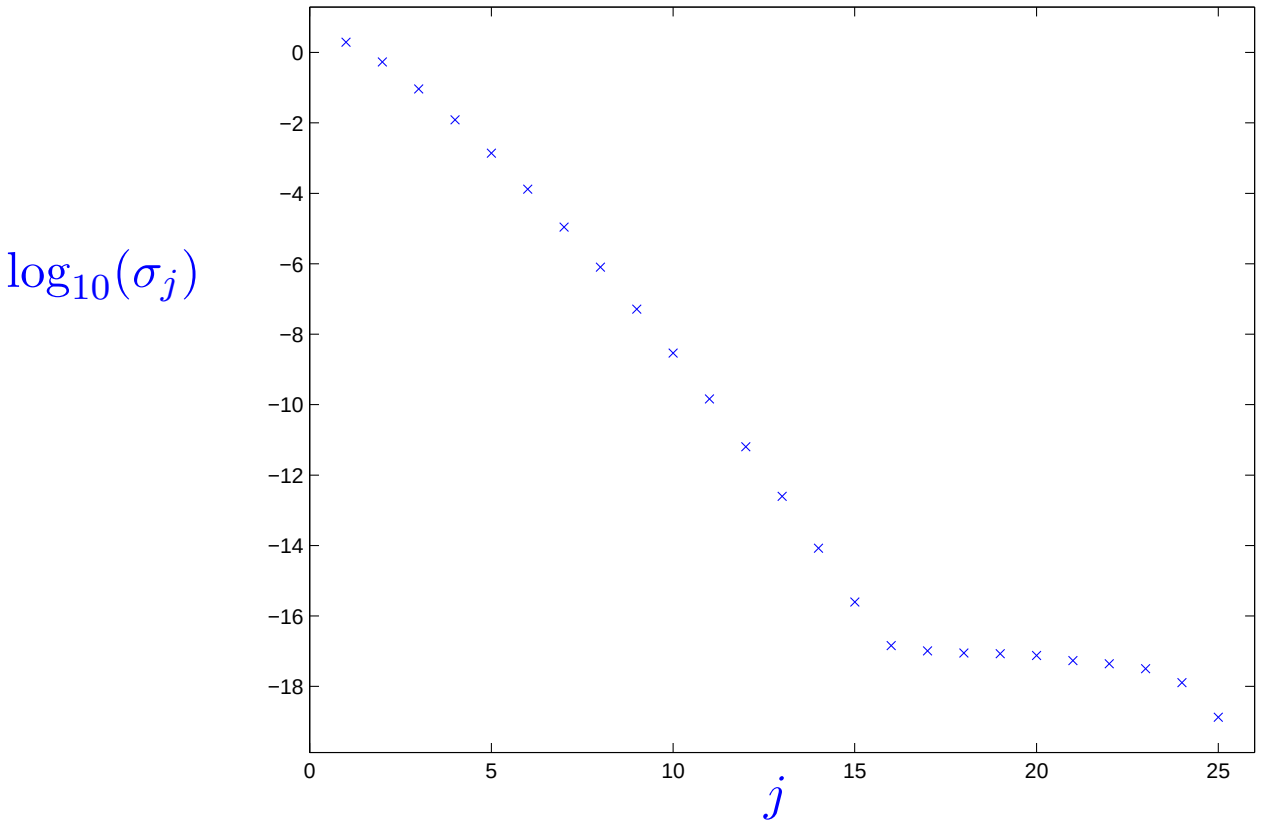
$$\sigma_j = \inf_{\mathrm{rank}(\tilde{A})=j-1} ||A - \tilde{A}||.$$

and

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

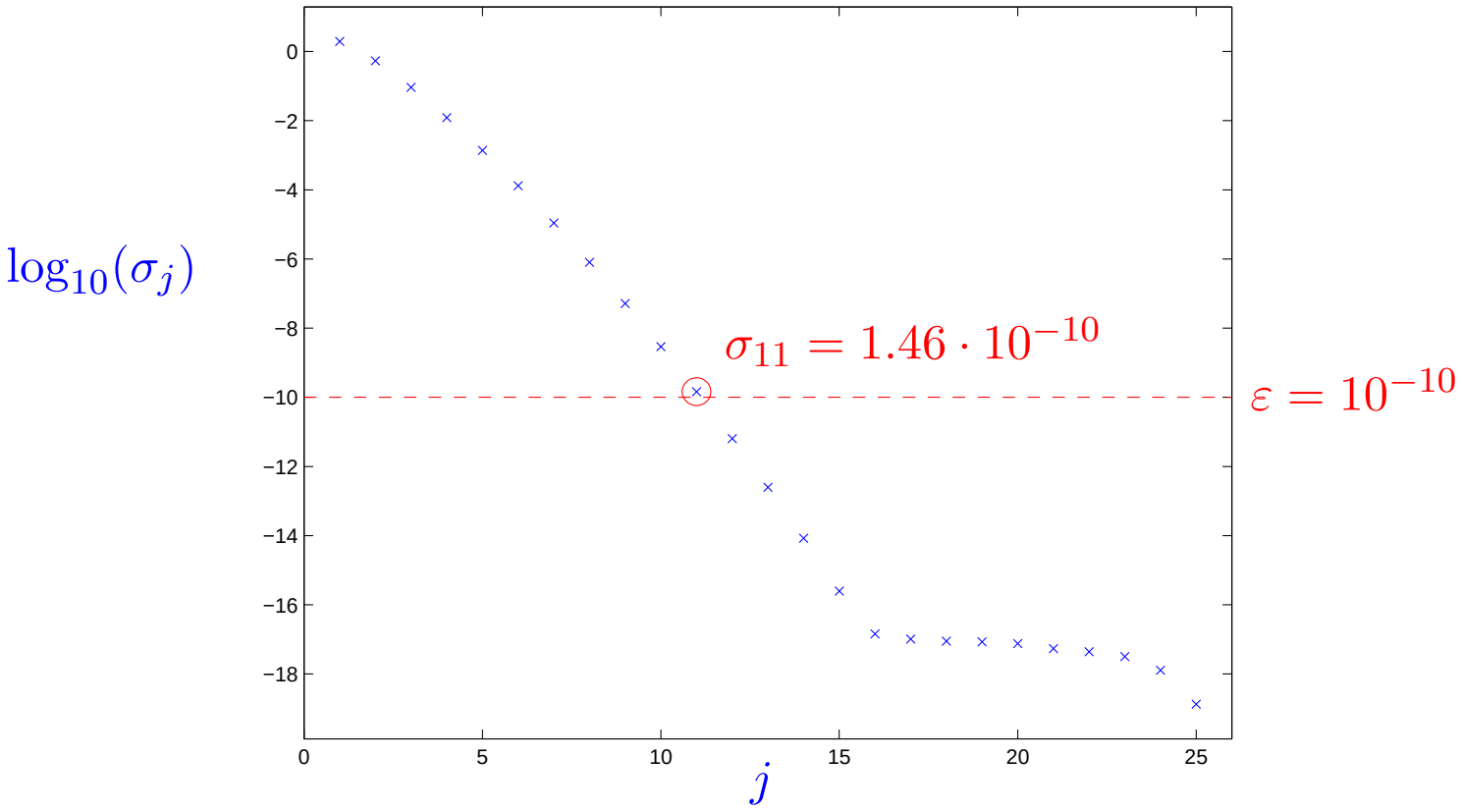
The decay of the singular values determines how well a matrix can be approximated by low-rank factorizations.

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.



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

Example: Let A be a 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 specified 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 $\mathrm{Col}(A)$. • 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 matrix $Y = A\,\Omega$.

3. Construct an ON 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

$$||A - Q\,Q^{\mathrm{t}}\,A|| \leq \varepsilon. \tag{1}$$

If we were very lucky, then (1) 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

$$e_\ell = ||A - Q_\ell\, Q_\ell^{\mathrm{t}}\, A||$$

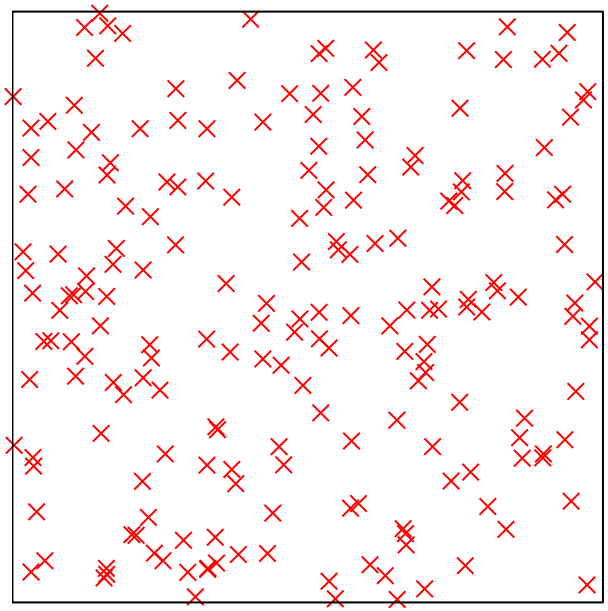
The quantity e_ℓ should be compared to the minimal error

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

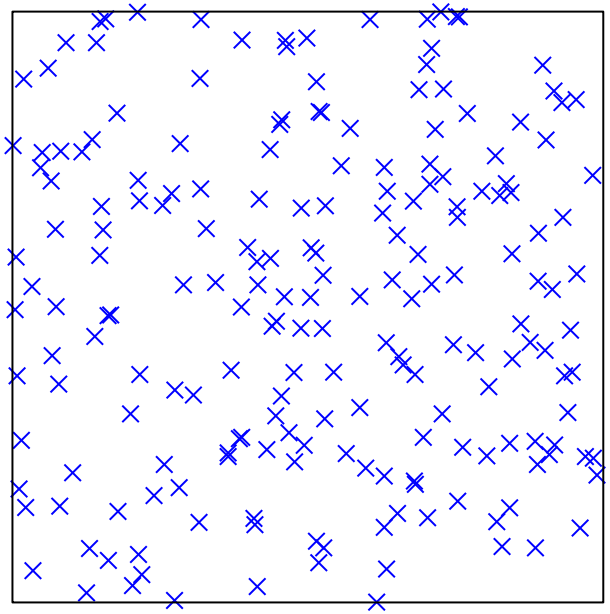
To illustrate the performance, we consider the following particular choice of A:

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

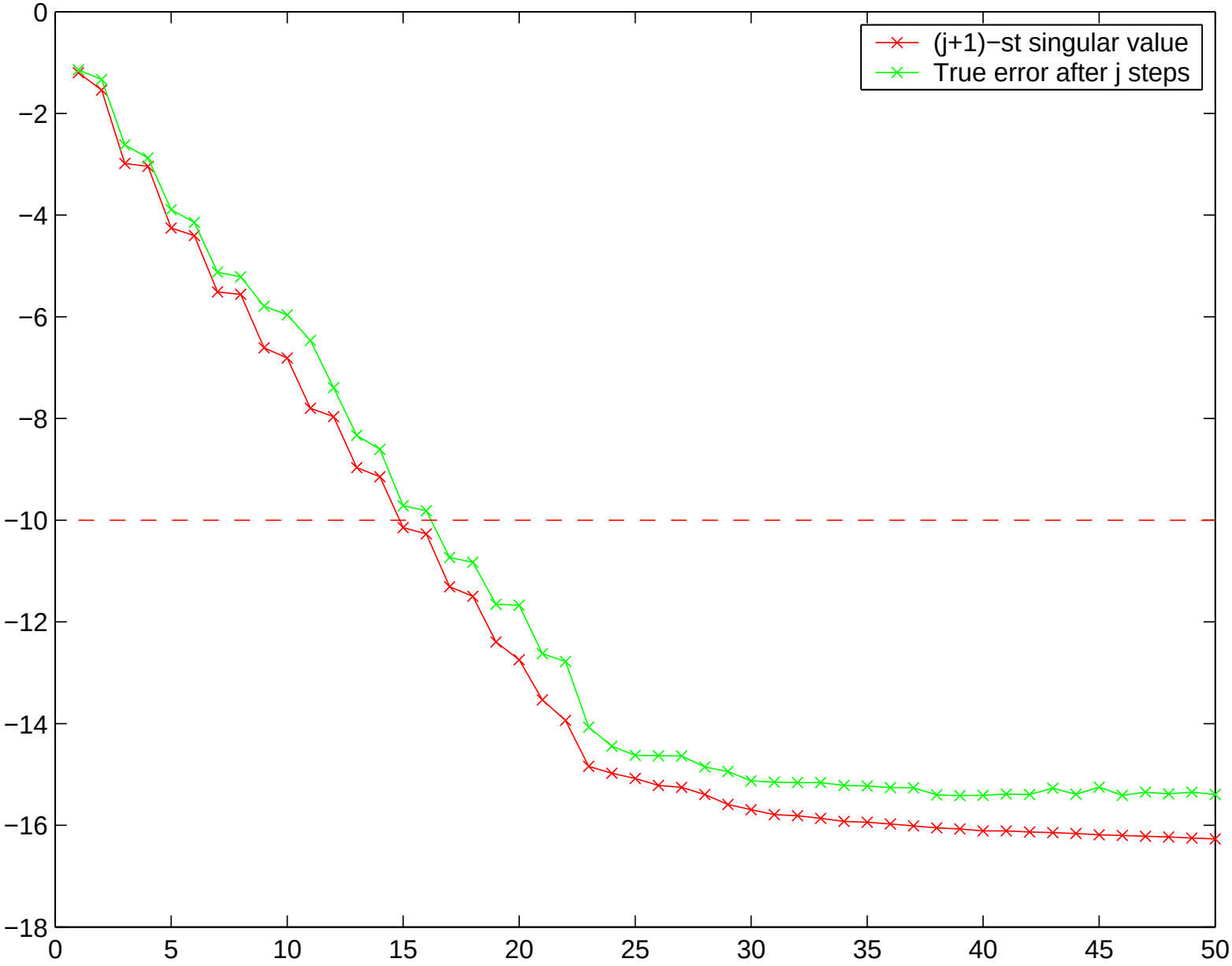
The number α is set so that $||A|| = 1$.



sources z_i



targets w_j



$\varepsilon = 10^{-10}$, exact ε-rank $= 15$, nr. of samples actually required $= 17$.

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

$$e_\ell = ||A - Q_\ell\, Q_\ell^{\mathrm{t}}\, A||$$

The quantity e_ℓ should be compared to the minimal error

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

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

$$e_\ell = ||A - Q_\ell\, Q_\ell^{\mathrm{t}}\, A||$$

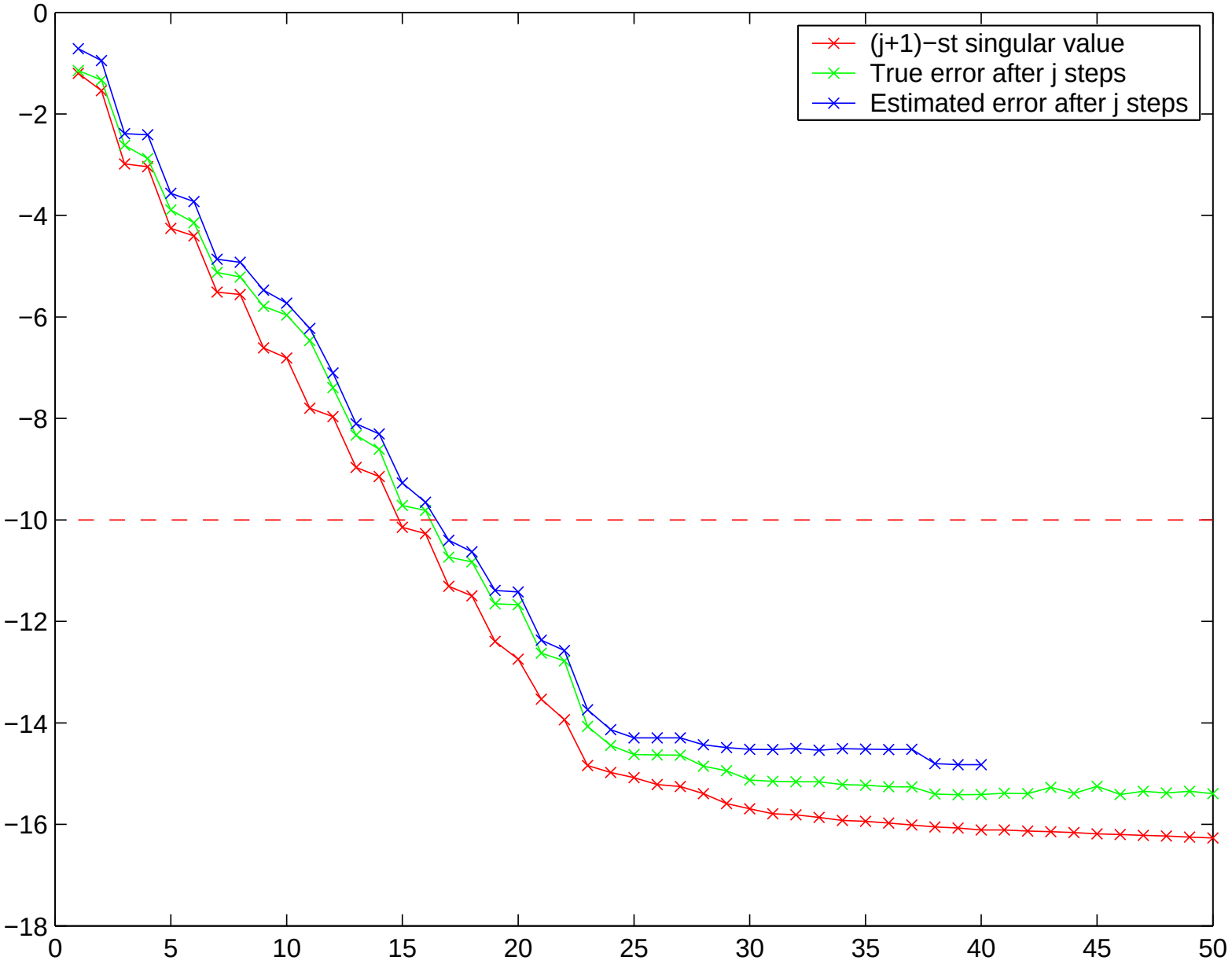
The quantity e_ℓ should be compared to the minimal error

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

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

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

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



$\varepsilon = 10^{-10}$, exact ε-rank $= 15$, estimated nr. of samples required $= 17{+}10$.

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

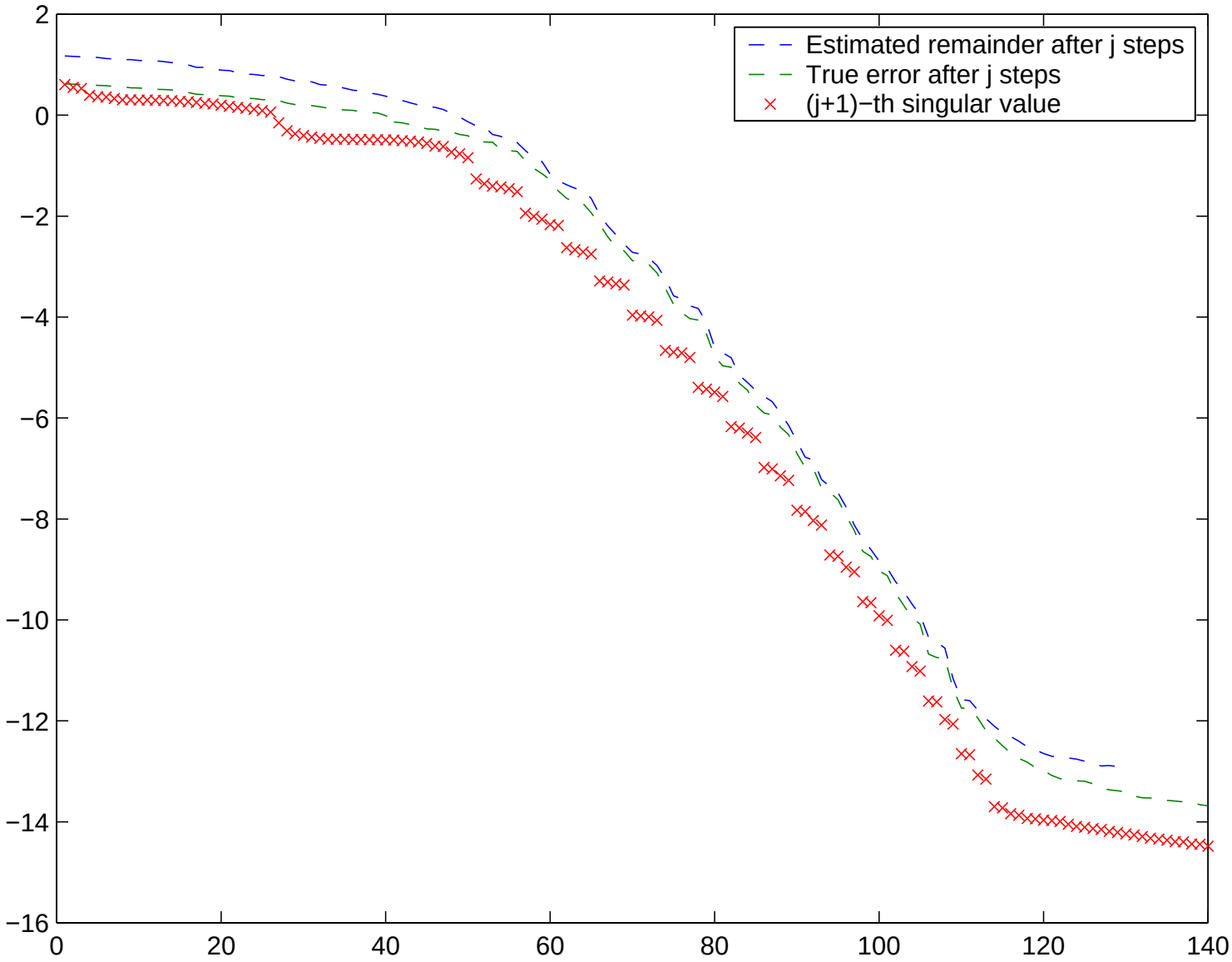
Was this just a lucky realization?

We collected statistics from 1 000 000 realizations:
(Recall that the theoretical minimal number is 15.)

Number of matrix-vector multiplies required:	Frequency:
16 (+10)	2248
17 (+10)	638173
18 (+10)	313600
19 (+10)	45028
20 (+10)	948
21 (+10)	3

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 to be 15 *every time*.

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



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

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

What happens if we do not?

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

$$\underset{N \times \ell}{Y} = \underset{N \times N}{A} \; \underset{N \times \ell}{\Omega}.$$

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 $\mathbb{C}$.

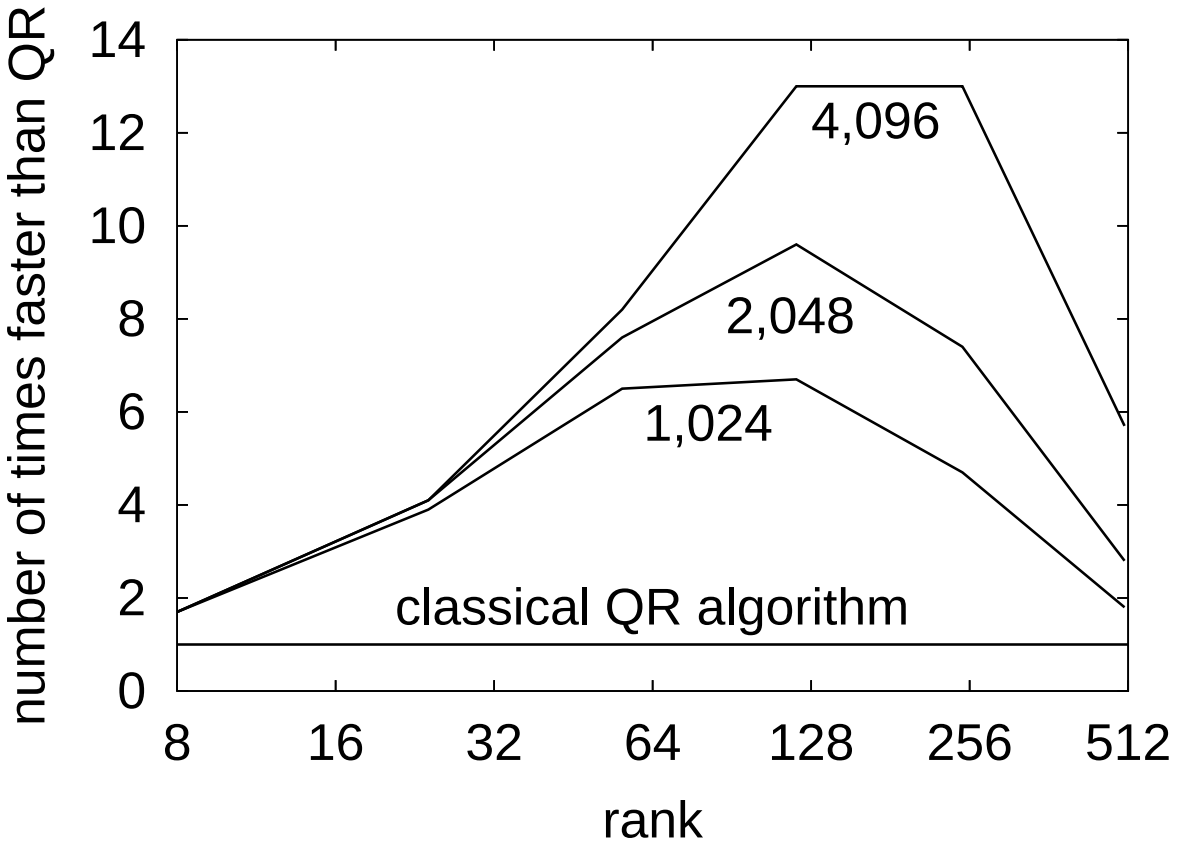
- F is the discrete Fourier transform, $F_{jk} = e^{-2\pi i(j-1)(k-1)/N}$.

- 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.

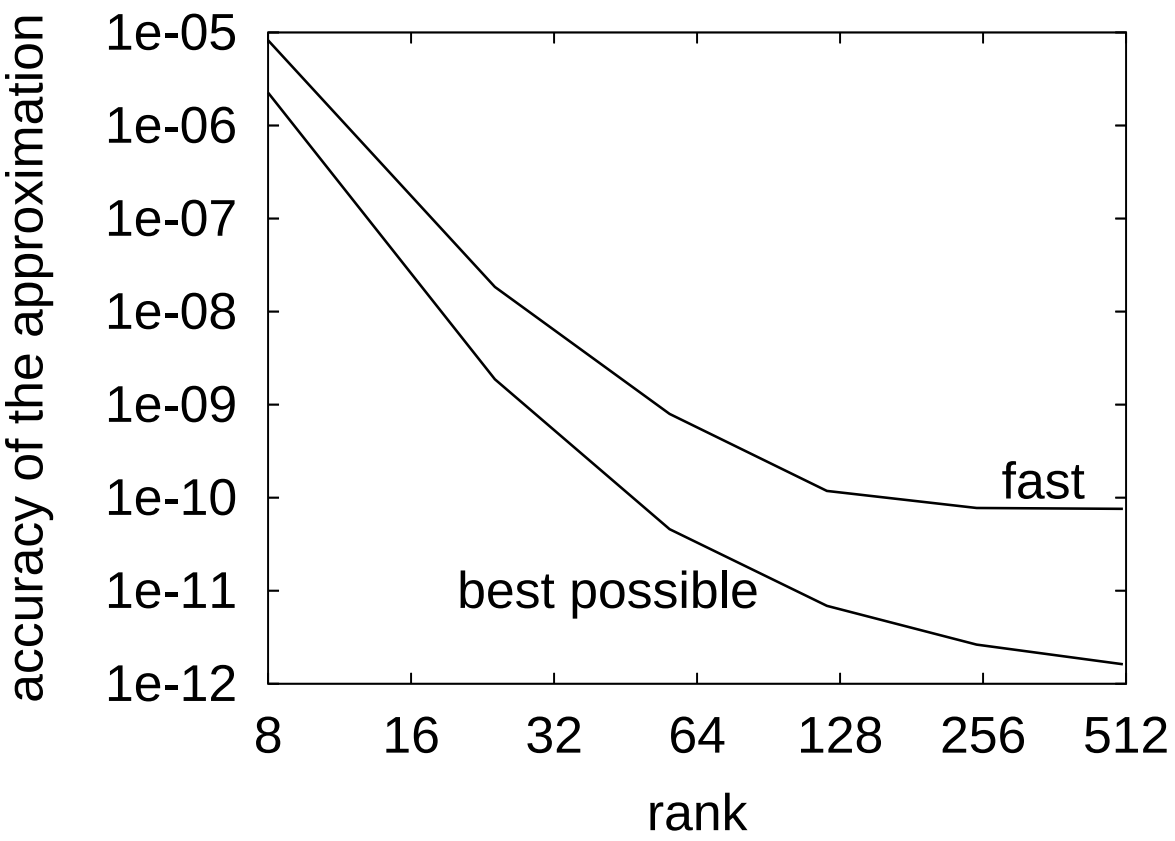
This idea was described by Nir Ailon and Bernard Chazelle (2006).
There is also related recent work by Sarlós (on randomized regression).

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

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

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.)

In applications where the singular values do not decay rapidly (as is typical for statistical data analysis, analysis of networks, *etc*), better estimates are needed.

Reducing the factor is a topic of active research — but it appears that in most environments, the factor can more or less entirely be eliminated.

- "A posteriori" analysis — Halko/Martinsson/Tropp — Sep. 2008.
- Use of power iterations — Rokhlin/Szlam/Tygert — Oct. 2008.
- Improved proofs — Halko/Martinsson/Tropp — Dec. 2008.
- That the "fixes" work has been demonstrated by examples with $N \sim 10^7$ and $\sigma_{k+1} \sim 10^{-2}$ — Martinsson/Tygert/Shkolnisky — Dec. 2008.

Notes on the proof (of the simplest version):

We assume without loss of generality that A is diagonal and square since:

- The probability distribution of a Gaussian matrix is invariant under rotations.
- All steps of the algorithm are also invariant under rotations.

Then

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

The question becomes: How large must ℓ be before we can say with high probability that the columns of an $N \times \ell$ 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" 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*

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

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

$$(1 - \varepsilon) \, ||u - v||^2 \leq ||f(u) - f(v)|| \leq (1 + \varepsilon) \, ||u - v||^2, \qquad \forall \; u, v \in V. \tag{3}$$

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, (2), this map satisfies (3) 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.

The randomized algorithms provide what is in many ways theoretically optimal methods for computing PCAs (and other approximate low-rank factorizations) to very large, very noisy matrices.

Work is under way to demonstrate the effectiveness of the techniques by applying them to problems arising in, *e.g.*:

- Genomics.
- Image processing.
- Network matrices ("page rank" type problems).

These techniques are also applicable to many "classical" applications:

- Acceleration of BLAS / LINPACK functions.
- Construction of reduced models for physical phenomena ("model reduction").
  - Wave propagation through media with periodic micro-structures.
  - Crack propagation through composite materials.
  - Scattering problems involving multiple scatterers.
- Acceleration of fast matrix algorithms such as the "Fast Multipole Method".

The randomized algorithms provide what is in many ways theoretically optimal methods for computing PCAs (and other approximate low-rank factorizations) to very large, very noisy matrices.

Work is under way to demonstrate the effectiveness of the techniques by applying them to problems arising in, *e.g.*:

- Genomics.
- Image processing.
- Network matrices ("page rank" type problems).

These techniques are also applicable to many "classical" applications:

- Acceleration of BLAS / LINPACK functions.
- Construction of reduced models for physical phenomena ("model reduction").
  - Wave propagation through media with periodic micro-structures.
  - Crack propagation through composite materials.
  - Scattering problems involving multiple scatterers.
- Acceleration of fast matrix algorithms such as the "Fast Multipole Method".

How can the randomized algorithms improve the Fast Multipole Method (FMM)?

First, we recall that the FMM is a fast method for evaluating sums such as

$$u_i = \sum_{j=1}^{N} G(x_i, x_j)\, q_j,$$

where G is a given kernel function defined on $\mathbb{R}^d \times \mathbb{R}^d$,
$\{x_i\}_{i=1}^N$ is a given set of points in $\mathbb{R}^d$,
$\{q_i\}_{i=1}^N$ is a given set of numbers ("charges"), and
$\{u_i\}_{i=1}^N$ is the set of numbers to be computed ("potentials").

In effect, we are computing the matrix-vector product

$$u = A\, q,$$

where A is the (typically dense) $N \times N$ matrix with entries

$$A_{ij} = G(x_i, x_j).$$

The idea behind the FMM is that many such matrices can be tesselated into submatrices that have low rank. By organizing the computation appropriately, it is then often possible to evaluate the product

$$q \mapsto A\,q,$$

in $O(N)$ operations rather than the $O(N^2)$ operations required by direct summation.

"Classical" fast methods depend in a strong sense on a known kernel.
The approximate factorizations are derived from analytic formulas.

What can classical Fast Multipole Methods do?

- Compute matrix-vector products?
  Yes!
- Solve linear systems?
  Yes — provided that iterative methods converge reasonably fast.
- Compute matrix inverses?
  No, the kernel of the inverse is not known.
- Compute matrix-matrix products?
  No, the kernel of the product is not known.
- Compute partial spectral decompositions?
  Sometimes — only when Lanczos/Arnoldi type methods work.

What can classical Fast Multipole Methods do?

Suppose that we use randomized sampling to factor the off-diagonal blocks.

- Compute matrix-vector products?
  Yes!
- Solve linear systems?
  Yes, directly!
- Compute matrix inverses?
  Yes.
- Compute matrix-matrix products?
  Yes.
- Compute partial spectral decompositions?
  Yes. Via inverse iteration any part of the spectrum can be retrieved.

(The storytelling is simplified (and tendentious!) — there do exist "direct" methods that do not rely on randomized sampling.)

So randomized sampling helps overcome the problem of an unknown kernel.

Unfortunately, existing methods for performing "fast" matrix algebra
are efficient only in low-dimensional geometry ($d = 1, 2, 3$).

The reason is that the tessellation of the matrix is derived from a hierarchical binning of the computational points. It gets complicated in $\mathbb{R}^2$, bad in $\mathbb{R}^3$, terrible beyond that ...

Outstanding challenge: How can we design fast methods for performing matrix algebra rapidly on operators defined on point clouds in high dimensional spaces that have some type of "structure"?

Summary

We have demonstrated a technique for approximately factoring matrices that have numerically low rank. The procedure is based on random sampling.

Its performance is very close to what is theoretically possible.

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:

- Optimal tool for computing PCAs of large noisy data matrices.
- Definitively overcome issues relating to stability and parallelization of Lanzcos.
- Reduction of CPU time requirement of partial spectral decompositions, QR factorizations, *etc*, from $O(N^2\,k)$ to $O(N^2\log(k))$.

There should be much much more to come.