(A randomized method for the approximation of matrices)

Two randomized methods for the approximation of matrices

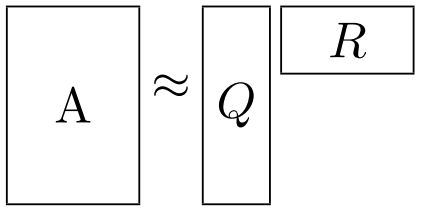
P.G. Martinsson, The University of Colorado at Boulder

Acknowledgements:

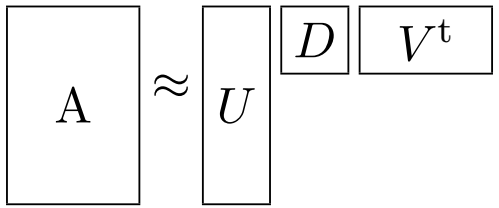
Some of the work presented is joint work with Vladimir Rokhlin and Mark Tygert.

Some of the work presented is work by Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert

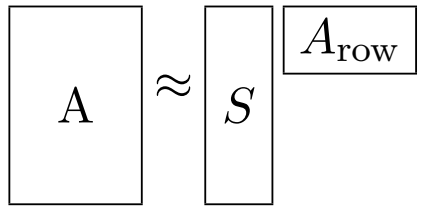
Let A be an $m \times n$ matrix that can be approximated by a matrix of rank k:



"QR-decomposition"



"SVD"



"Interpolative decomposition"

Question: How do you efficiently find such approximations?

A classical answer: Compute the QR-factorization via, *e.g.* Gram-Schmidt. Cost is $O(mnk)$.

Algorithm 1: When matrix-vector products $x \mapsto A\,x$ can be computed cheaply, say at a cost T_{mult}, the cost can be reduced to $O(T_{\text{mult}}\,k + m\,k^2)$.

Algorithm 2: When A is a general matrix (not necessarily cheap to apply), the cost can be reduced to $O(m\,n\,\log(k) + (m+n)\,k^2)$.

Algorithms 1 and 2 are based on randomized methods, meaning that they have a probability of failure. This probability is typically negligible (like 10^{-17}).

Related work on randomized algorithms:

- Dixon (1983)
- Wozniakowski and Kuczynsky (1993)
- 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)

We will "declare victory" once we have a basis for the column space.

To justify this, suppose that we have found an orthonormal matrix Q such that

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

Then at a cost of $O(m\, k^2)$ we determine k rows of Q that form a well-conditioned basis for the row-space of Q. Collecting these into Q_{row}, we have

$$Q = P \begin{bmatrix} I_k \\ T \end{bmatrix} Q_{\mathrm{row}},$$

where P is a permutation matrix. Then

$$A \approx Q\, Q^{\mathrm{t}}\, A = P \begin{bmatrix} I_k \\ T \end{bmatrix} Q_{\mathrm{row}}\, Q^{\mathrm{t}}\, A = P \begin{bmatrix} I_k \\ T \end{bmatrix} A_{\mathrm{row}},$$

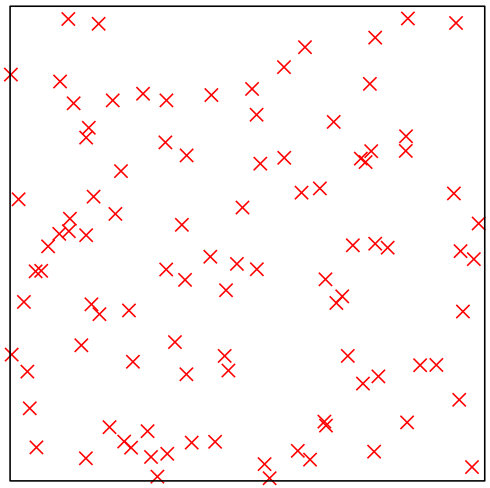
where A_{row} consists of k rows of A.

So, once Q is determined, only an $O(k^2\, m)$, or possibly $O(k^2\, (m+n))$, cost remains.

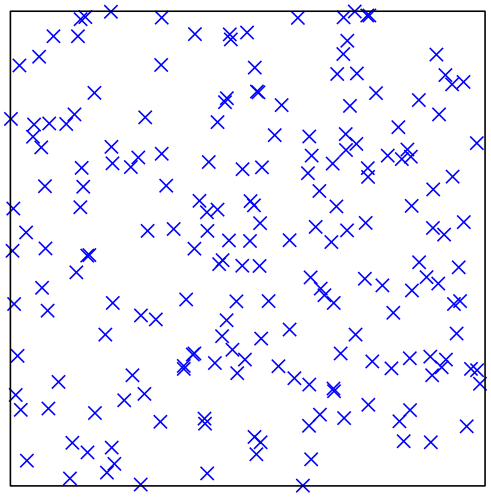
Algorithm 1

For when the matrix-vector multiplication $x \mapsto A\,x$ is cheap.

Examp



Ω_{S}



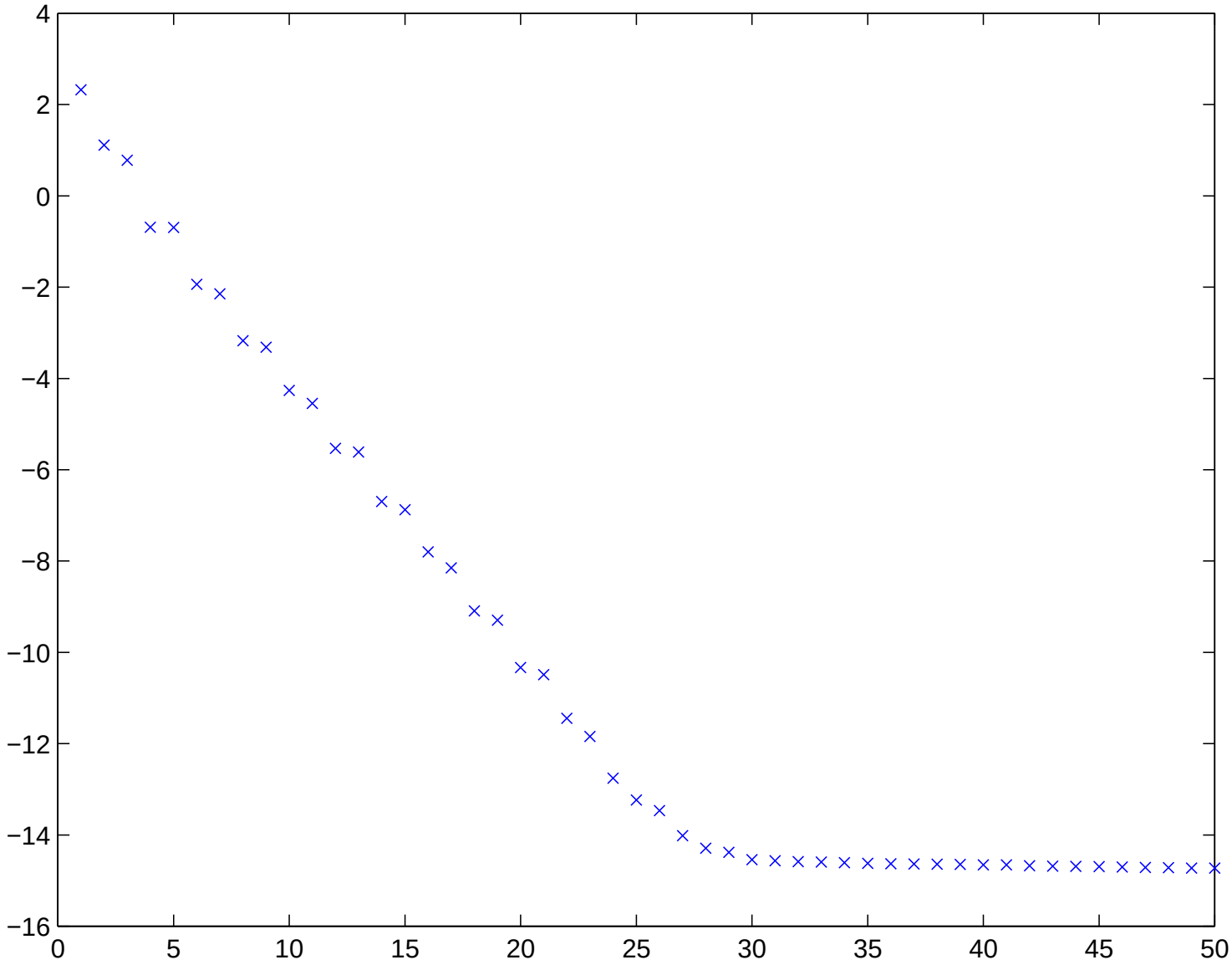
Ω_{T}

Points $\{w_j\}_{j=1}^{n}$ in Ω_{S} ("sources").

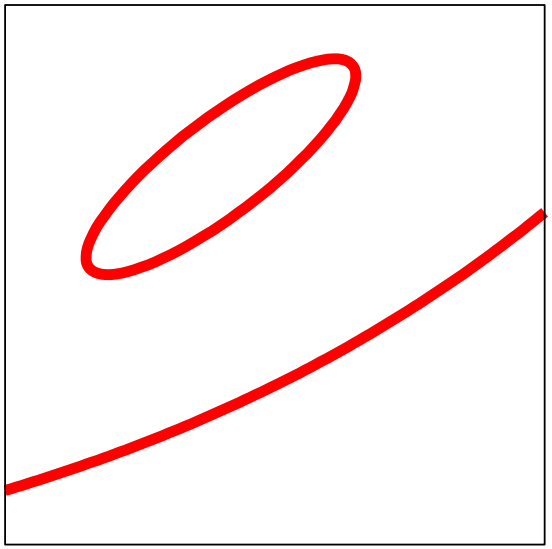
Points $\{z_i\}_{i=1}^{m}$ in Ω_{T} ("targets").

Let A be an $m \times n$ matrix with entries $A_{ij} = \log|z_i - w_j|$.

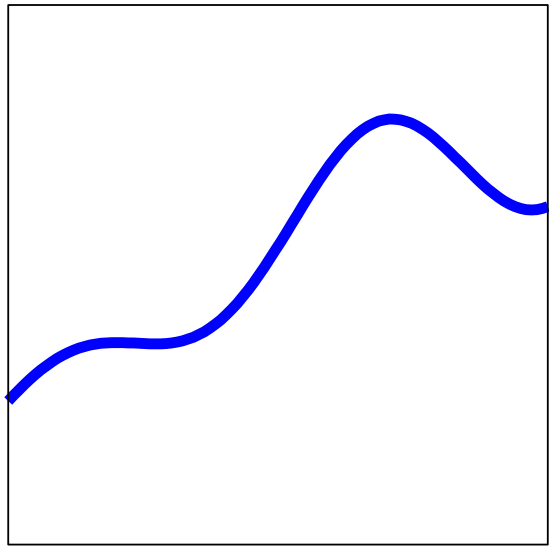
"A maps a charge distribution to a set of potentials."



The 10-logarithm of the singular values of A.



Γ_{S}



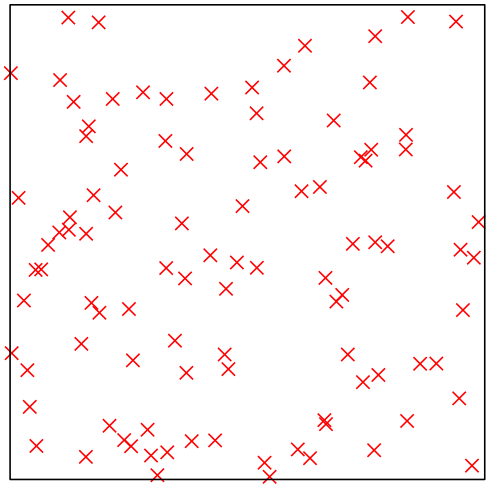
Γ_{T}

The same type of spectrum is obtained for the "off-diagonal blocks" of many integral operators:

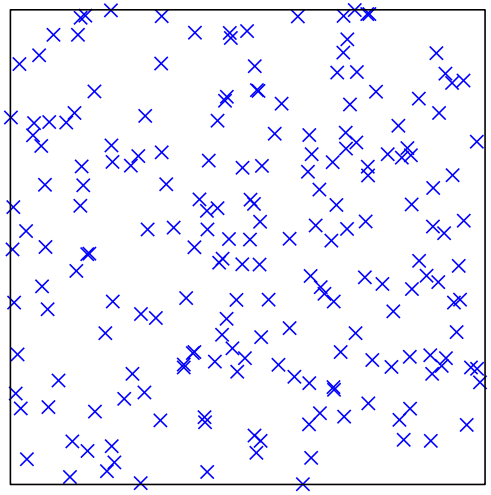
$$[A\,u](x) = \int_{\Gamma_{\mathrm{s}}} G(x,y)\,u(y)\,ds(y), \qquad x \in \Gamma_{\mathrm{T}}.$$

For instance, G could be the single or double layer kernel for the Laplace equation.

Examp



Ω_{S}



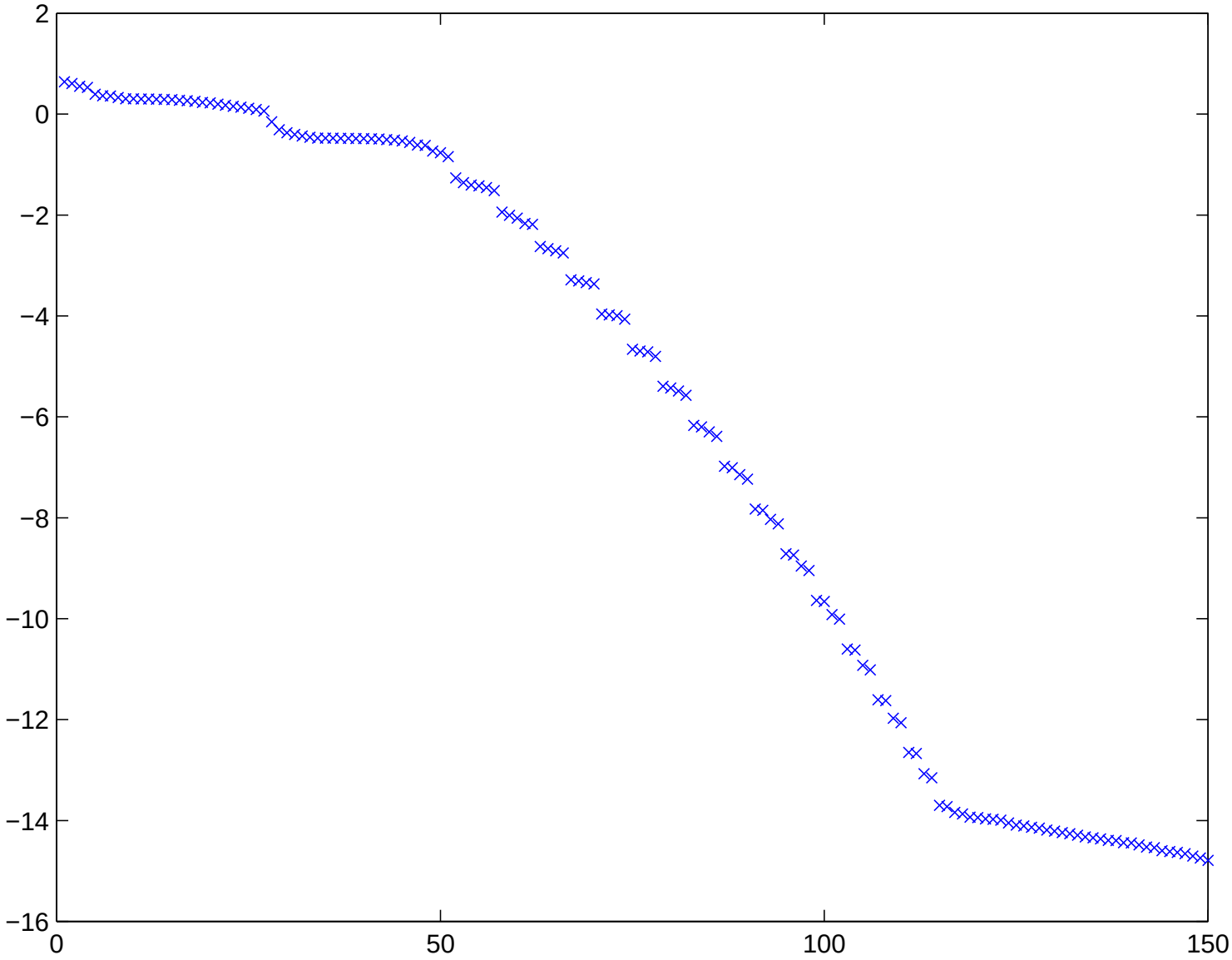
Ω_{T}

Points $\{w_j\}_{j=1}^{n}$ in Ω_{S} ("sources").

Points $\{z_i\}_{i=1}^{m}$ in Ω_{T} ("targets").

Let A be an $m \times n$ matrix with entries $A_{ij} = H_0^{(1)}(k\,|z_i - w_j|)$.

"A maps a charge distribution to a set of potentials."



The 10-logarithm of the singular values of A for $k = 35$.

Example:

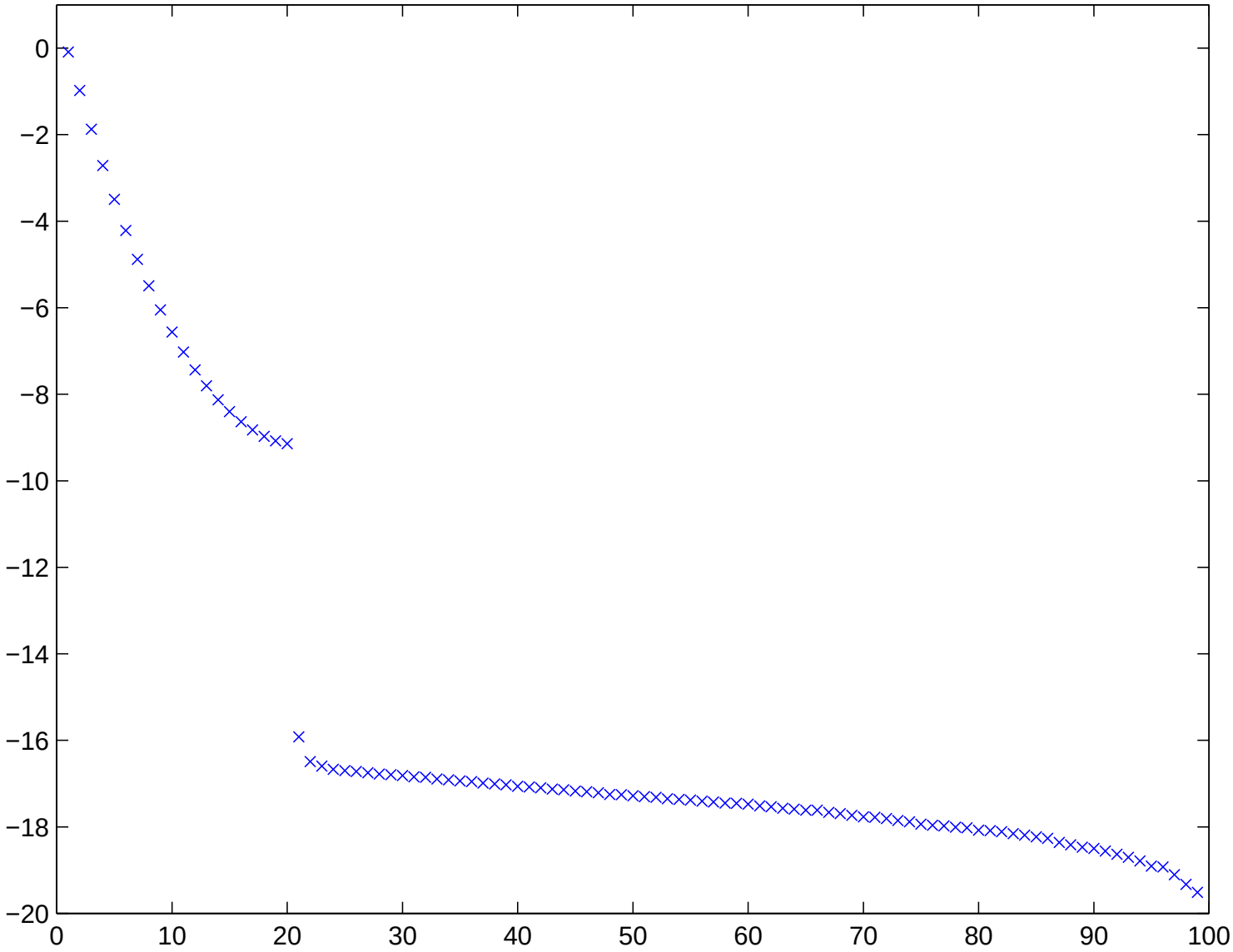
Let B be the standard five-point stencil on a 20×20 grid:

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

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

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

We consider the 100×100 submatrix A_{14} of the 400×400 matrix A.



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

Algorithm 1 — for when we can compute $x \mapsto A\,x$ rapidly

Recall: A is $m \times n$ with ε-rank k.

Let $x_1, x_2, \ldots$ be a sequence of vectors in $\mathbb{R}^n$ whose entries are i.i.d. random variables drawn from a standardized Gaussian distribution. (Alternatively, draw them uniformly from the surface of the unit ball.)

Form the length-m vectors

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

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.

Algorithm 1 — for when we can compute $x \mapsto A\,x$ rapidly

Recall: A is $m \times n$ with ε-rank k.

Let $x_1, x_2, \ldots$ be a sequence of vectors in $\mathbb{R}^n$ whose entries are i.i.d. random variables drawn from a standardized Gaussian distribution. (Alternatively, draw them uniformly from the surface of the unit ball.)

Form the length-m vectors

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

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

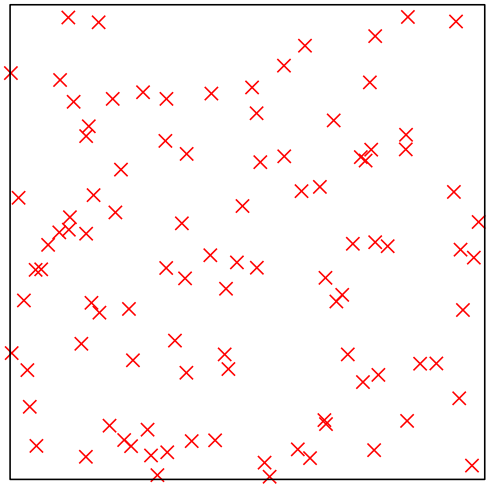
$$\{y_1, y_2, \ldots, y_l\}$$

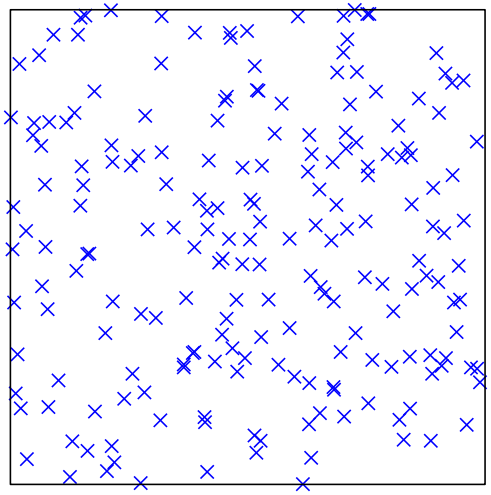
span the column space of A "to within precision ε". Clearly, the probability that this happens gets larger, the larger the gap between l and k.

What is remarkable is how fast this probability approaches one.

We illustrate with a numerical example.

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





Generate a sequence $x_1, x_2, \ldots$ of random vectors in $\mathbb{R}^n$.

Compute $Y_l = [y_1, y_2, \ldots, y_l] = [A\,x_1, A\,x_2, \ldots, A\,x_l]$.

Compute the (column pivoted) QR-factorization $Y_l = Q_l\,R_l\,P_l$.

The "error" after l steps is (using the l^2-operator norm)

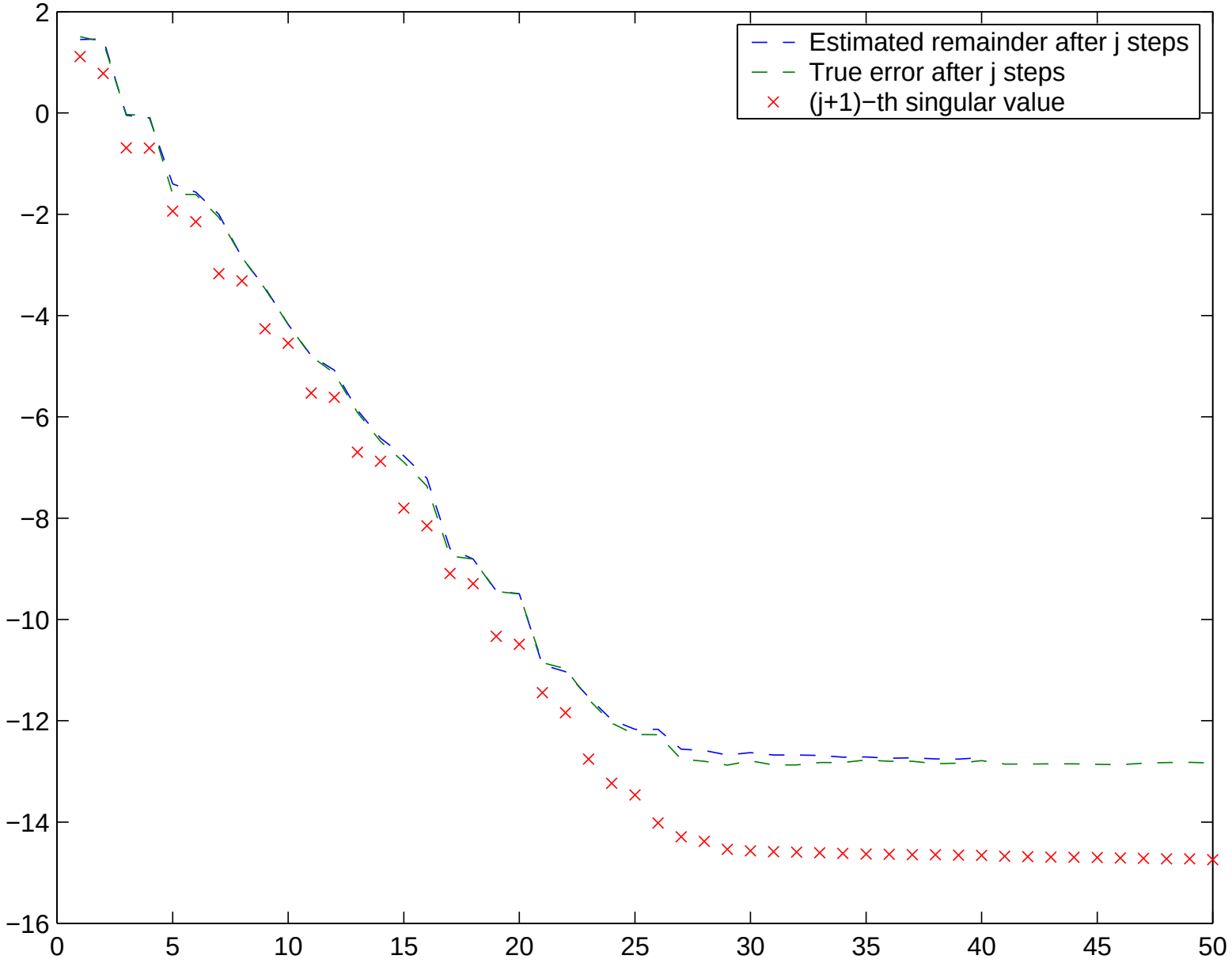
$$e_l = ||(I - Q_l\,Q_l^{\mathrm{t}})\,A||.$$

Notice that in reality, we can rarely afford to compute e_l.
Instead, we compute something like

$$f_l = \left|\left|(I - Q_l\,Q_l^{\mathrm{t}})\,[y_{l+1}, y_{l+2}, \cdots, y_{l+10}]\frac{1}{10}\right|\right|_{\text{Frobenius}}.$$

Our estimate for the rank is the lowest integer l such that $f_l < \varepsilon$.

(Notice that plenty of operations here can be optimized. A lot.)



$\varepsilon = 10^{-10}$ True ε-rank $= 19$ Estimated ε-rank $= 21$ / 19.

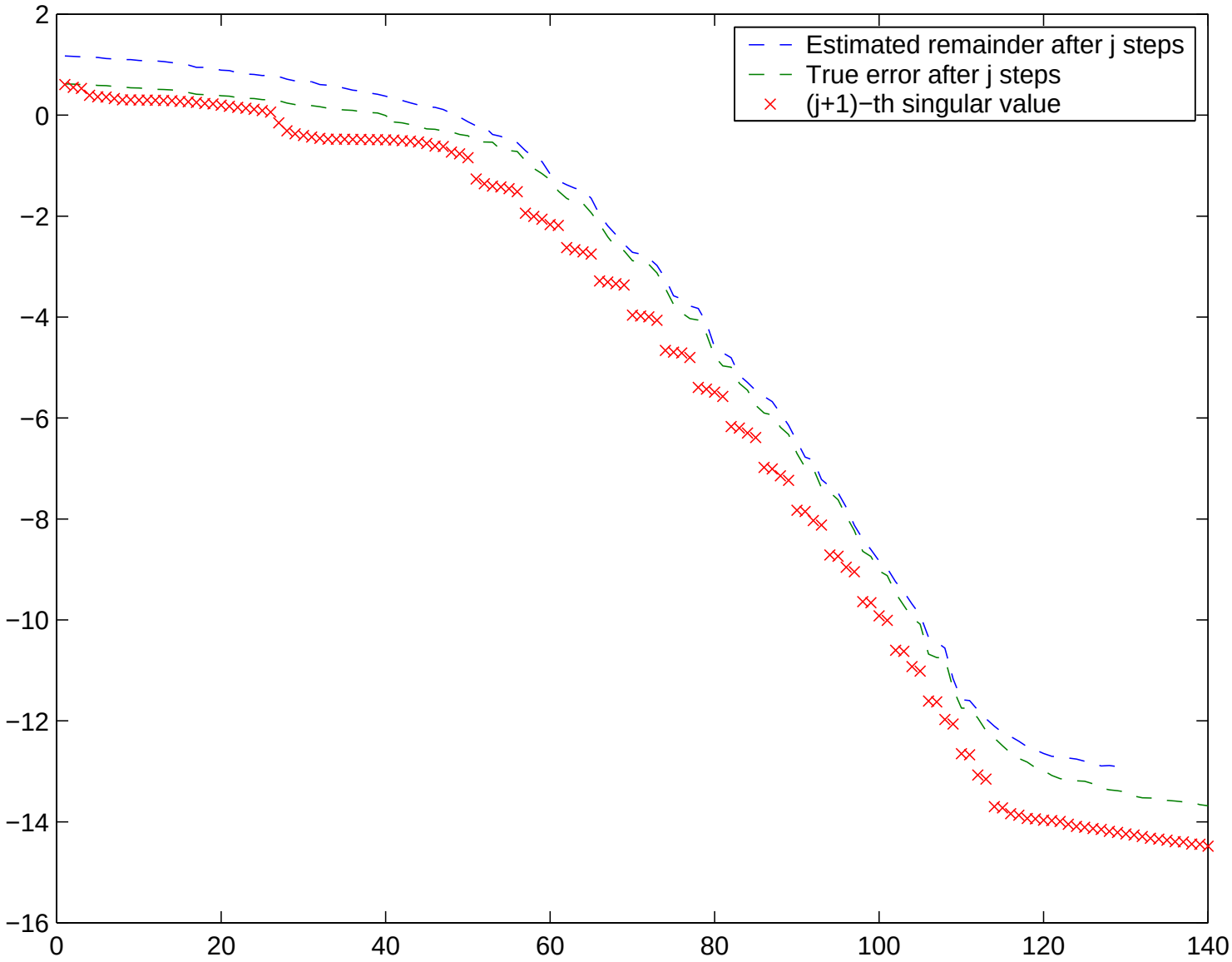
Was this just a lucky realization?

We ran the algorithm a million times and got these estimated ranks:

```
k = 17:          0 times
k = 18:          0 times
k = 19:       4178 times
k = 20:     246905 times
k = 21:     664486 times
k = 22:      81789 times
k = 23:       2634 times
k = 24:          8 times
k = 25:          0 times
k = 26:          0 times
```

The numbers above relate to the *initial estimate* of the rank. The second estimate was *always* 19, and the error was *always* less than 10^{-10}.

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



$\varepsilon = 10^{-10}$ True ε-rank $= 101$ Estimated ε-rank $= 106$ / 101.

Theorem: *Let A be an $m \times n$ matrix and let k be an integer.*

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

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

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

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

Then

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

with probability at least

$$1 - f(l-k),$$

where f is a decreasing function satisfying

$$f(8) < 10^{-5}$$

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

Recall the error bound:

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

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

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

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

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

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

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

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

Other potential benefits:

- Data-movement.
- Parallelization.

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

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

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

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

$$\underset{m \times l}{Y} \quad = \quad \underset{m \times n}{A} \quad \underset{n \times l}{G}.$$

Key points:

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

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

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

$$a \mapsto a\,G$$

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

What is this "random but structured" matrix G?

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

where

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

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

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

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

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

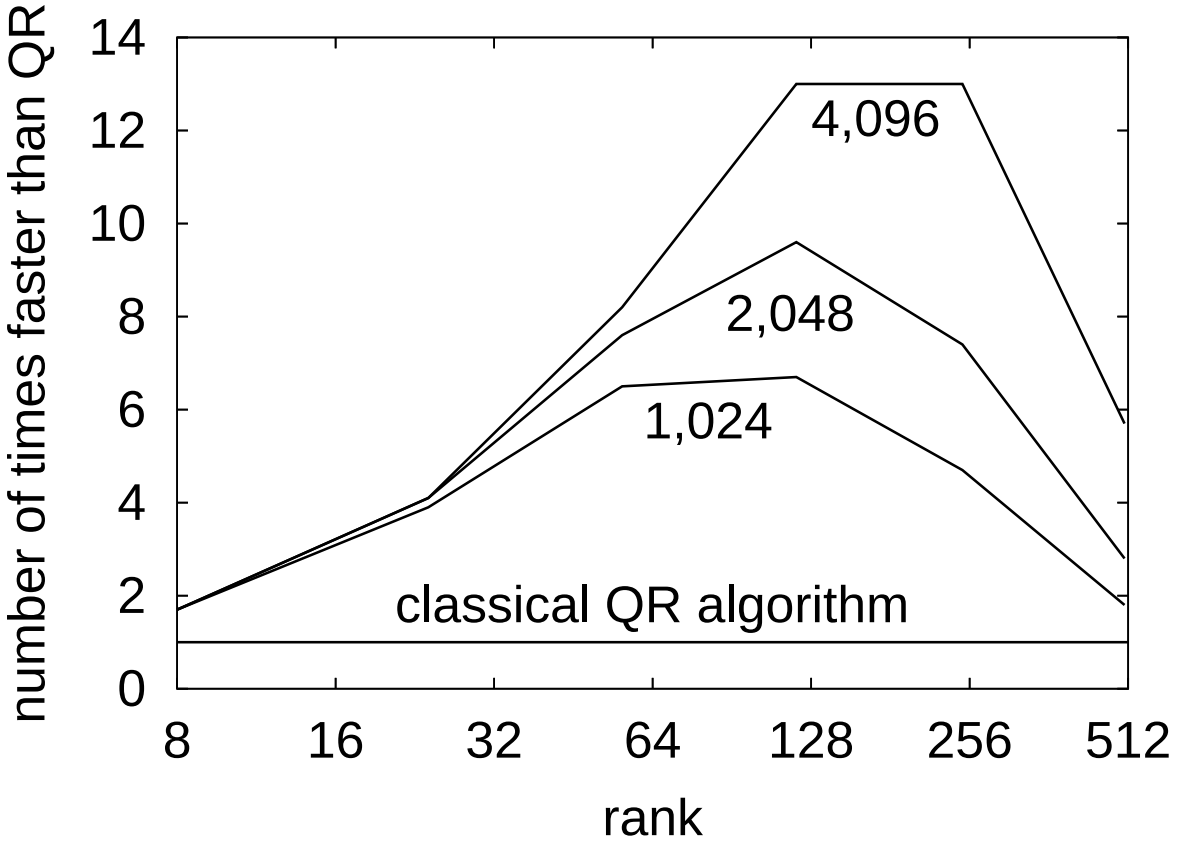
What is the probability of failure?

The proofs obtained so far do not assure quite as high likelihood of success as the proofs for Algorithm 1 did. (Say $1 - 10^{-7}$ instead of $1 - 10^{-17}$.)

The proofs may not be sharp however. An indication that this may be the case is that the algorithm has never failed during testing.

Should it prove to be the case that Algorithm 2 occasionally fails, a cheap verification can be put in place. (Simply note that the difference between A and the computed approximation to A can rapidly be applied to a vector.)

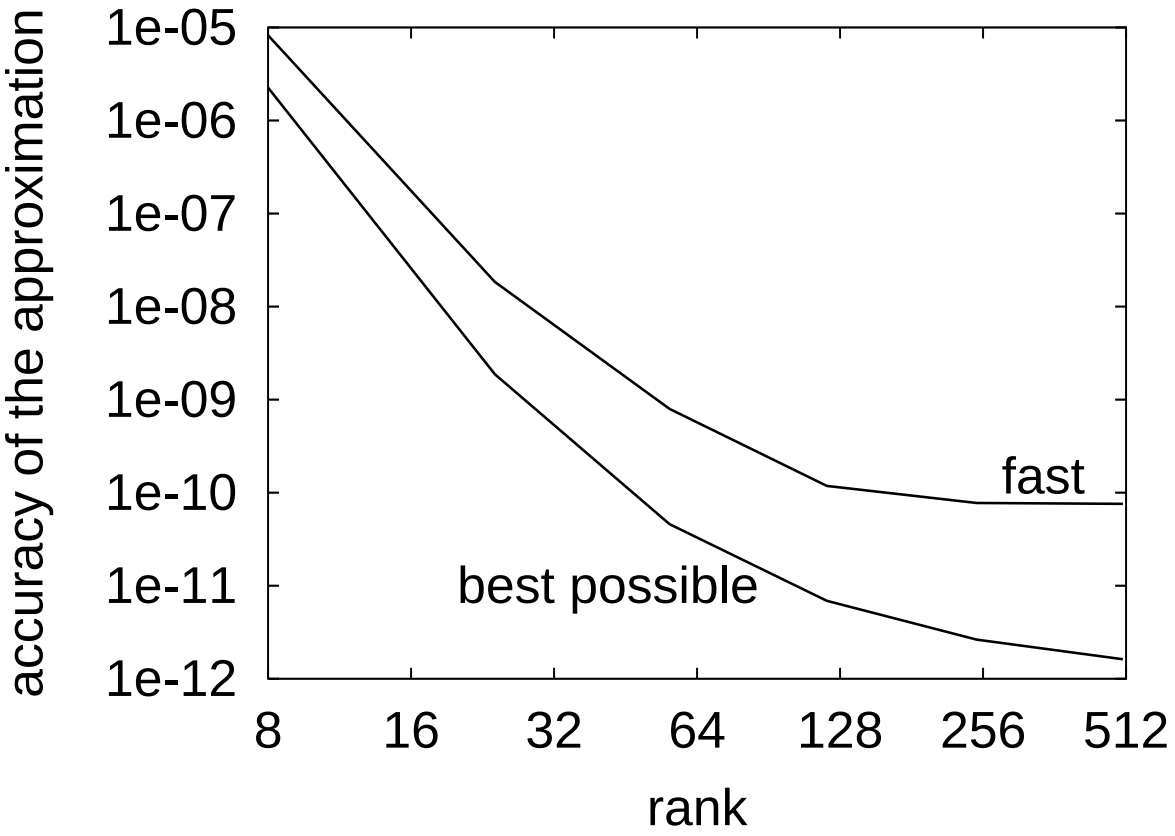
Speed gain on square matrices of various sizes



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

This slide comes from a talk by Mark Tygert.

Empirical accuracy on 2,048-long convolution



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

This slide comes from a talk by Mark Tygert.

Key points — Algorithm 2:

(Franco Woolfe, Edo Liberty, Vladimir Rokhlin, Mark Tygert)

There exists an algorithm for rank-k matrix approximation (or for computing the top k singular values and vectors) with advantages over the classical pivoted $Q\,R$ algorithms such as Gram-Schmidt:

1. Substantially faster (for most ranks k of the approximation), costing $O(n^2 \ln(k) + nk^2)$ — not $O(n^2 k)$ — for an $n \times n$ matrix.

2. Uses less storage when the input matrix is to be preserved, especially for matrices evaluated on-the-fly.

3. Operates reliably and accurately on any matrix.

4. Parallelizes naturally.

Future work:

- Develop efficient strategies for determining the rank adaptively, and for updating the ON-basis for the column space.
- Develop ways to decrease the probability of failure in Algorithm 2.
- Tighten the proofs for Algorithm 2.
- Check if the algorithms can be modified to improve the factors "$\sqrt{ml}$" in the error bounds.

Applications:

- Fast algorithms for matrix algebra (matrix-vector multiplies, matrix-inversions, spectral decompositions) involving differential and integral operators.
- Multiscale modelling.
- Analysis of network matrices (data mining).