Randomized Algorithms for Very Large-Scale Linear Algebra

Gunnar Martinsson
The University of Colorado at Boulder

Collaborators: Edo Liberty, Vladimir Rokhlin (congrats!), Yoel Shkolnisky, Arthur Szlam, Joel Tropp, Mark Tygert, Franco Woolfe, ...

Student: Nathan Halko (now at Spot Influence, LLC).

Paper: N. Halko, P.G. Martinsson, J. Tropp. *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.* SIAM Review, **53**(2), 2011.

Slides: Posted under the "Talks" tab on my webpage (google "Gunnar Martinsson").

Objective:

Given an $n \times n$ matrix **A**, for a very large n (say $n \sim 10^6$),
we seek to compute a rank-k approximation, with $k \ll n$ (say $k \sim 10^2$ or 10^3),

$$\underset{n \times n}{\mathbf{A}} \approx \underset{n \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}^*} = \sum_{j=1}^{k} \mathbf{e}_j \, \mathbf{f}_j^*.$$

Solving this problem leads to algorithms for computing:

- Eigenvectors corresponding to leading eigenvalues.
  (Require $\mathbf{e}_j = \lambda_j \, \mathbf{f}_j$, and $\{\mathbf{f}_j\}_{j=1}^k$ to be orthonormal.)
- Singular Value Decomposition (SVD) / Principal Component Analysis (PCA).
  (Require $\{\mathbf{e}_j\}_{j=1}^k$ and $\{\mathbf{f}_j\}_{j=1}^k$ to be orthogonal sets.)
- Spanning columns or rows.
  (Require $\{\mathbf{e}_j\}_{j=1}^k$ to be columns of **A**, or require $\{\mathbf{f}_j^*\}_{j=1}^k$ to be rows of **A**.)
- *etc*

The problem being addressed is ubiquitous in applications.

Note: The only error that will be controlled is the "backwards error" $||\mathbf{A} - \mathbf{EF}|| \leq$ TOL.

Applications:

- Fast algorithms for elliptic PDEs: more efficient Fast Multipole Methods, fast *direct* solvers, construction of special quadratures for corners and edges, etc.

- Statistical analysis via Principal Component Analysis.

- Data mining (machine learning, analysis of network matrices, etc).

- Diffusion geometry; a technique for constructing parameterizations on large collections of data points organized (modulo noise) along non-linear low-dimensional manifolds. Requires the computations of eigenvectors of *graph Laplace operators*.

- Nearest neighbor search for large clouds of points in high dimensional space.

- General pre-conditioners.

- Etc.

Review of existing methods I

For a dense $n \times n$ matrix that fits in RAM, excellent algorithms are already part of LAPACK (and Matlab, *etc*).

- Double precision accuracy.
- Very stable.
- $O(n^3)$ asymptotic complexity. Reasonably small constants.
- Require extensive random access to the matrix.

When the target rank k is much smaller than n, there also exist $O(n^2\,k)$ methods with similar characteristics (the well-known Golub-Businger method, RRQR by Gu and Eisentstat, *etc*).

For small[a] matrices the state-of-the-art is very satisfactory.
For kicks, we will improve on it anyway, but this is not the main point.

[a] On a standard 2012 personal computer, "small" means $n \leq 10\,000$ or so.

Review of existing methods II

If the matrix is large, but can rapidly be applied to a vector (if it is sparse, or sparse in Fourier space, or amenable to the FMM, etc.), so called *Krylov subspace methods* often yield excellent accuracy and speed.

The idea is to pick a starting vector ω (often a random vector), "restrict" the matrix $\mathbf{A}$ to the k-dimensionsal "Krylov subspace"

$$\text{Span}(\omega,\, \mathbf{A}\,\omega,\, \mathbf{A}^2\,\omega,\, \ldots,\, \mathbf{A}^{k-1}\,\omega)$$

and compute approximate eigenvectors of the resulting matrix. Advantages:

- Very simple access to $\mathbf{A}$.
- Extremely high accuracy possible.

Drawbacks:

- In standard implementations, the matrix is revisited $O(k)$ times if a rank-k approximation is sought. (Blocked versions exist, but the convergence analysis is less developed.)
- There are numerical stability issues. These are well-studied and can be overcome, but they make the algorithms less portable (between applications, hardware platforms, etc.).

"New" challenges in algorithmic design:

The existing state-of-the-art methods of numerical linear algebra that we have very briefly outlined were designed for an environment where the matrix fits in RAM and the key to performance was to minimize the number of floating point operations required.

Currently, *communication* is becoming the real bottleneck:

- While clock speed is hardly improving at all anymore, the cost of a flop keeps going down rapidly. (Multi-core processors, GPUs, cloud computing, etc.)

- The cost of slow storage (hard drives, flash memory, etc.) is also going down rapidly.

- Communication costs are decreasing, but *not* rapidly.
  - Moving data from a hard-drive.
  - Moving data between nodes of a parallel machine. (Or cloud computer ...)
  - The amount of fast cache memory close to a processor is not improving much. (In fact, it could be said to be *shrinking* — GPUs, multi-core, etc.)

- "Deluge of data". Driven by ever cheaper storage and acquisition techniques. Web search, data mining in archives of documents or photos, hyper-spectral imagery, social networks, gene arrays, proteomics data, sensor networks, financial transactions, ...

The more powerful computing machinery becomes,
the more important efficient algorithm design becomes.

- Linear scaling (w.r.t. problem size, processors, etc.).
- Minimal data movement.

That *randomization* can be used to overcome some of the communication bottlenecks in matrix computations has been pointed out by several authors:

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, *etc*)

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)

K. Clarkson, D. Woodruff (2009)

Literature survey: Halko, Martinsson, Tropp (2011).

Review of existing methods III

Examples of how randomization could be used:

- **Random column/row selection**
  Draw at random some columns and suppose that they span the entire column space.
  If rows are drawn as well, then spectral properties can be estimated.
  Crude sampling leads to less than $O(mn)$ complexity, but is very dangerous.

- **Sparsification**
  Zero out the vast majority of the entries of the matrix. Keep a random subset of entries, and boost their magnitude to preserve "something."

- **Quantization and sparsification**
  Restrict the entries of the matrix to a small set of values (-1/0/1 for instance).

The methods outlined can be as fast as you like, but must necessarily have very weak performance guarantees. They can work well for certain classes of matrices for which additional information is available (basically, matrices that are in some sense "over-sampled").

Approach advocated here:

A randomized algorithm for computing a rank-k approximation to an $m \times n$ matrix.
It is engineered from the ground up to:

- Minimize communication.
- Handle streaming data, or data stored "out-of-core."
- Easily adapt to a broad range of distributed computing architectures.

Computational profile:

- At least $O(mn)$ complexity (in fact $O(mnk)$ or $O(mn \log(k))$).
- The accuracy ε is a user-set number.
  (If the application permits, it could be $\varepsilon = 10^{-12}$ or less.)
- Since the method is randomized, it has a *failure probability* η.
  η is a user specified number.
  The cost of the method grows as $\eta \rightarrow 0$, but setting $\eta = 10^{-10}$ is cheap.
  For all practical purposes, the methods succeed with probability 1.

Goal: Given an $m \times n$ matrix $\mathbf{A}$, compute an approximate rank-k SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

Goal: Given an $m \times n$ matrix $\mathbf{A}$, compute an approximate rank-k SVD $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$.

Algorithm:

1.
2.
3.

Find an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\mathbf{A} \approx \mathbf{Q}\,\mathbf{Q}^*\,\mathbf{A}$.
(I.e., the columns of $\mathbf{Q}$ form an ON-basis for the range of $\mathbf{A}$.)

4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\,\mathbf{A}$.
5. Compute the SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\,\mathbf{\Sigma}\,\mathbf{V}^*$.
6. Form the matrix $\mathbf{U} = \mathbf{Q}\,\hat{\mathbf{U}}$.

Goal: Given an $m \times n$ matrix $\mathbf{A}$, compute an approximate rank-k SVD $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$.

Algorithm:

1. *Find an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\mathbf{A} \approx \mathbf{Q}\,\mathbf{Q}^*\,\mathbf{A}$.*
2. *(I.e., the columns of $\mathbf{Q}$ form an ON-basis for the range of $\mathbf{A}$.)*
3.
4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\,\mathbf{A}$.
5. Compute the SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\,\mathbf{\Sigma}\,\mathbf{V}^*$.
6. Form the matrix $\mathbf{U} = \mathbf{Q}\,\hat{\mathbf{U}}$.

Note: Steps 4 – 6 are exact; the error in the method is all in $\mathbf{Q}$:

$$||\mathbf{A} - \underbrace{\mathbf{U}}_{=\mathbf{Q}\hat{\mathbf{U}}}\mathbf{\Sigma}\mathbf{V}^*|| = ||\mathbf{A} - \mathbf{Q}\underbrace{\hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^*}_{=\mathbf{B}}|| = ||\mathbf{A} - \mathbf{Q}\underbrace{\mathbf{B}}_{\mathbf{Q}^*\mathbf{A}}|| = ||\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}||.$$

Note: The classical *Golub-Businger algorithm* follows this pattern. It finds $\mathbf{Q}$ in Step 3 via direct orthogonalization of the columns of $\mathbf{A}$ via, e.g., Gram-Schmidt.

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

Solving the primitive problem via randomized sampling — intuition:

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

2. Form "sample" vectors $\mathbf{y}_1 = \mathbf{A}\,\boldsymbol{\omega}_1, \mathbf{y}_2 = \mathbf{A}\,\boldsymbol{\omega}_2, \ldots, \mathbf{y}_k = \mathbf{A}\,\boldsymbol{\omega}_k \in \mathbb{R}^m$.

3. Form orthonormal vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k \in \mathbb{R}^m$ such that

$$\text{Span}(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k) = \text{Span}(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k).$$

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

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

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

Solving the primitive problem via randomized sampling — intuition:

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

2. Form "sample" vectors $\mathbf{y}_1 = \mathbf{A}\,\boldsymbol{\omega}_1, \mathbf{y}_2 = \mathbf{A}\,\boldsymbol{\omega}_2, \ldots, \mathbf{y}_k = \mathbf{A}\,\boldsymbol{\omega}_k \in \mathbb{R}^m$.

3. Form orthonormal vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k \in \mathbb{R}^m$ such that

$$\text{Span}(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k) = \text{Span}(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k).$$

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

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

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

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

Solving the primitive problem via randomized sampling — intuition:

1. Draw a random matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times k}$.
   (We will discuss the choice of distribution later — think Gaussian for now.)

2. Form a "sample" matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega} \in \mathbb{R}^{m \times k}$.

3. Form an orthonormal matrix $\mathbf{Q} \in \mathbb{R}^{m \times k}$ such that $\mathbf{Y} = \mathbf{Q}\mathbf{R}$.
   For instance, Gram-Schmidt can be used — pivoting is rarely required.

If $\mathbf{A}$ has *exact* rank k, then $\mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$ with probability 1.

Goal: Given an $m \times n$ matrix $\mathbf{A}$, compute an approximate rank-k SVD $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$.

Algorithm:

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$.
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega}$.
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\mathbf{Y} = \mathbf{Q}\,\mathbf{R}$.
4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\,\mathbf{A}$.
5. Compute the SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\,\mathbf{\Sigma}\,\mathbf{V}^*$.
6. Form the matrix $\mathbf{U} = \mathbf{Q}\,\hat{\mathbf{U}}$.

Goal: Given an $m \times n$ matrix $\mathbf{A}$, compute an approximate rank-k SVD $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$.

Algorithm:

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,l)`
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega}$. `Y = A * Omega`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\mathbf{Y} = \mathbf{Q}\,\mathbf{R}$. `[Q, R] = qr(Y)`
4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\,\mathbf{A}$. `B = Q' * A`
5. Compute the SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\,\mathbf{\Sigma}\,\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B)`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\,\hat{\mathbf{U}}$. `U = Q * Uhat`

Single pass algorithms:

A is symmetric:	**A is not symmetric:**
Generate a random matrix $\mathbf{\Omega}$.	Generate random matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$.
Compute a sample matrix $\mathbf{Y}$.	Compute sample matrices $\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\,\mathbf{\Psi}$.
Find an ON matrix $\mathbf{Q}$ such that $\mathbf{Y} = \mathbf{Q}\,\mathbf{Q}^*\,\mathbf{Y}$.	Find ON matrices $\mathbf{Q}$ and $\mathbf{W}$ such that $\mathbf{Y} = \mathbf{Q}\,\mathbf{Q}^*\,\mathbf{Y}$ and $\mathbf{Z} = \mathbf{W}\,\mathbf{W}^*\,\mathbf{Z}$.
Solve for $\mathbf{T}$ the linear system $\mathbf{Q}^*\,\mathbf{Y} = \mathbf{T}\,(\mathbf{Q}^*\,\mathbf{\Omega})$.	Solve for $\mathbf{T}$ the linear systems $\mathbf{Q}^*\,\mathbf{Y} = \mathbf{T}\,(\mathbf{W}^*\,\mathbf{\Omega})$ and $\mathbf{W}^*\,\mathbf{Z} = \mathbf{T}^*\,(\mathbf{Q}^*\,\mathbf{\Psi})$.
Factor $\mathbf{T}$ so that $\mathbf{T} = \hat{\mathbf{U}}\,\mathbf{\Lambda}\,\hat{\mathbf{U}}^*$.	Factor $\mathbf{T}$ so that $\mathbf{T} = \hat{\mathbf{U}}\,\mathbf{\Sigma}\,\hat{\mathbf{V}}^*$.
Form $\mathbf{U} = \mathbf{Q}\,\hat{\mathbf{U}}$.	Form $\mathbf{U} = \mathbf{Q}\,\hat{\mathbf{U}}$ and $\mathbf{V} = \mathbf{W}\,\hat{\mathbf{V}}$.
Output: $\mathbf{A} \approx \mathbf{U}\mathbf{\Lambda}\mathbf{U}^*$	Output: $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$

Note: With $\mathbf{B}$ as on the previous slide we have $\mathbf{T} \approx \mathbf{B}\,\mathbf{Q}$ (sym. case) and $\mathbf{T} \approx \mathbf{B}\,\mathbf{W}$ (nonsym. case).

Note: An alternative procedure based on "interpolative decompositions" is often more accurate.

References: *Woolfe, Liberty, Rokhlin, and Tygert (2008), Clarkson and Woodruff (2009), HMT2009, HMT2011.*

Note: If **A** has exact rank k, then the algorithm produces an exact answer with probability 1.

At least three "issues" need attention:

1. In practice, **A** does not have exact rank k.
   There are tail singular modes that pollute the answer.

2. In practice, the rank of **A** is not known in advance.
   Instead, we are typically given a precision ε, and need to find the rank.

3. Computations will be carried out in finite precision arithmetic.
   Since the basis is by nature badly conditioned, care has to be exercised. But:
   - If standard procedures of numerical linear algebra are used (correctly...),
     things work out fine.
   - However, one must only ask reasonable things of the algorithm:
     Good question: Find $\mathbf{U}_{\text{approx}}$, $\mathbf{\Sigma}_{\text{approx}}$, $\mathbf{V}_{\text{approx}}$ s.t. $||\mathbf{A} - \mathbf{U}_{\text{approx}}\mathbf{\Sigma}_{\text{approx}}\mathbf{V}_{\text{approx}}|| \leq \varepsilon$.
     Bad question: Find $\mathbf{U}_{\text{approx}}$, $\mathbf{\Sigma}_{\text{approx}}$, $\mathbf{V}_{\text{approx}}$ such that $||\mathbf{U}_{\text{exact}} - \mathbf{U}_{\text{approx}}|| \leq \varepsilon$, etc.

We will in this talk henceforth ignore question 3 (in the interest of not propagating misconceptions about what numerical analysts find interesting).

Input: An $m \times n$ matrix $\mathbf{A}$ and a target rank k.

Output: Rank-k factors $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question:

How much does the computation above cost?

Answer:

Cost of steps 2 and 4: Application of $\mathbf{A}$ and $\mathbf{A}^*$ to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

We will consider three proto-typical environments.

Input: An $m \times n$ matrix **A** and a target rank k.

Output: Rank-k factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Cost of steps 2 and 4: Application of **A** and $\mathbf{A}^*$ to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

Case 1: **A** is presented as an array of real numbers in RAM.

Cost is dominated by the $2\,m\,n\,k$ flops required for steps 2 and 4.

The $O(m\,n\,k)$ flop count is the same as that of standard methods such as Golub-Businger. However, the algorithm above requires no random access to the matrix **A** — the data is retrieved *in two passes.* (One pass in the modified version.)

Remark 1: Improved data access leads to a moderate gain ($\times 2$) in execution time.

Remark 2: Using a special structured random sampling matrix (for instance, a "subsampled random Fourier transform" / "fast Johnson-Lindenstrauss transform"), substantial gain in execution time, and asymptotic complexity of $O(m\,n\,\log(k))$ can be achieved.

Input: An $m \times n$ matrix **A** and a target rank k.

Output: Rank-k factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Cost of steps 2 and 4: Application of **A** and $\mathbf{A}^*$ to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

Case 1: **A** is presented as an array of real numbers in RAM.

Use a *subsampled random Fourier Transform (SRFT)*

$$\begin{array}{ccccc} \mathbf{\Omega} & = & \mathbf{D} & \mathbf{F} & \mathbf{S} \\ n \times k & & n \times n & n \times n & n \times k \end{array}$$

- **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, $\mathbf{F}_{pq} = \frac{1}{\sqrt{n}} e^{-2\pi i (p-1)(q-1)/n}$.
- **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 k columns at random from **D F**.)

References: Ailon and Chazelle (2006); Liberty, Rokhlin, Tygert, and Woolfe (2006).

Practical speed of the SRFT-accelerated method

Consider the task of computing a rank-k SVD of a matrix **A** of size $n \times n$.

$t^{(\mathrm{direct})}$ Time for classical (Golub-Businger) method — $O(k\,n^2)$

$t^{(\mathrm{srft})}$ Time for randomized method with an SRFT — $O(\log(k)\,n^2)$

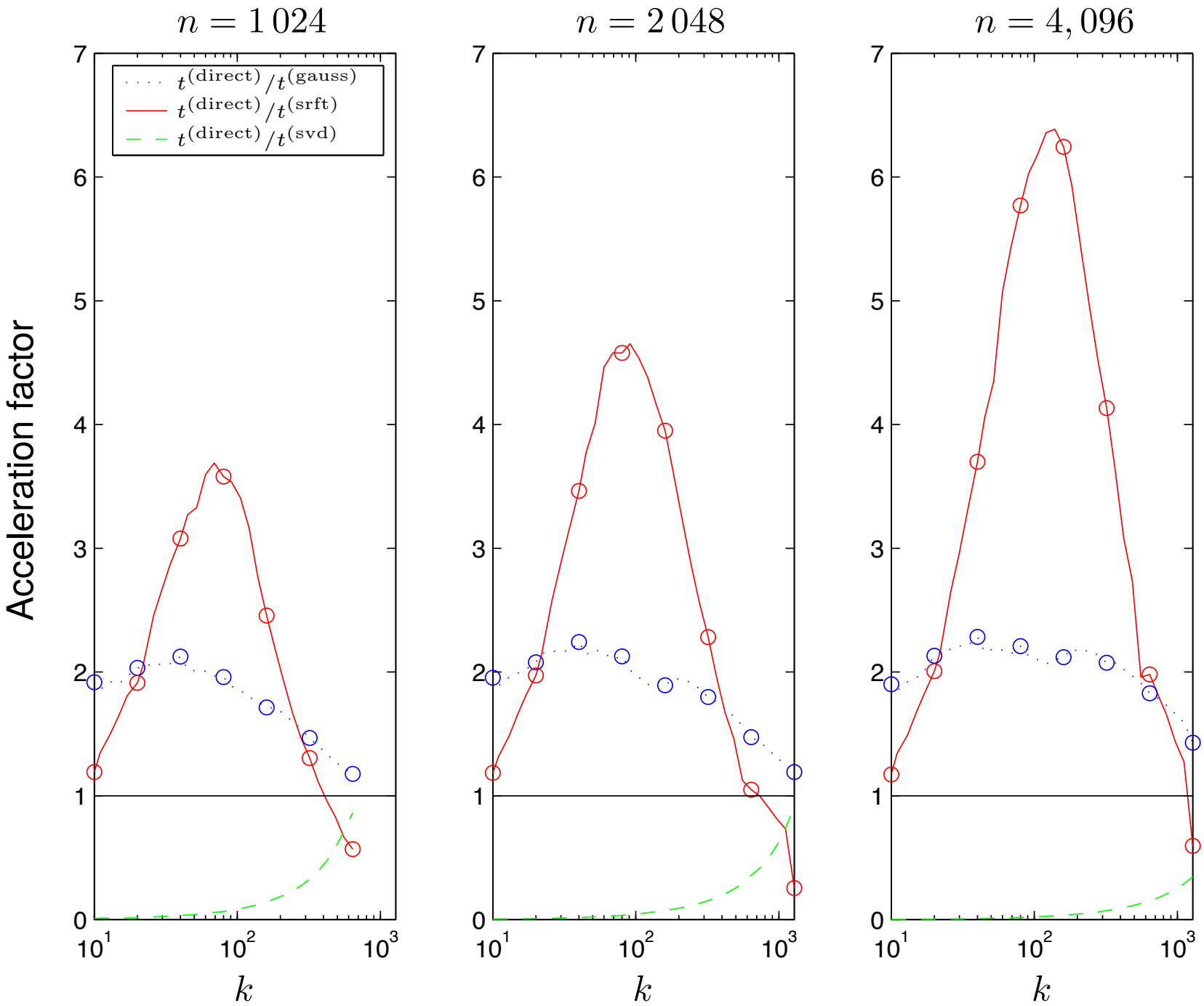
$t^{(\mathrm{gauss})}$ Time for randomized method with a Gaussian matrix — $O(k\,n^2)$

$t^{(\mathrm{svd})}$ Time for a full SVD — $O(n^3)$

We will show the

$$\text{acceleration factor} = \frac{t^{(\mathrm{direct})}}{t^{(\mathrm{srft})}}$$

for different values of n and k.


$n = 1\,024$
$n = 2\,048$
$n = 4,096$
Acceleration factor
$t^{(\mathrm{direct})}/t^{(\mathrm{gauss})}$
$t^{(\mathrm{direct})}/t^{(\mathrm{srft})}$
$t^{(\mathrm{direct})}/t^{(\mathrm{svd})}$
0
1
2
3
4
5
6
7
10^1
10^2
10^3
k

Notes:

- Significant speed-ups are achieved for common problem sizes. For instance, $m = n = 2\,000$ and $k = 200$ leads to a speed-up by roughly a factor of 4.

- Many other choices of random matrices have been found. (Subsampled Hadamard transform; wavelets; random chains of Given's rotations, etc).

- Using a randomly drawn selection of columns as a basis for the range of a matrix **A** is fine, *provided the matrix has first been hit by a random rotation.*

- Current theoretical results seem to overstate the risk of inaccurate results, at least in typical environments.

- The idea was proposed by Liberty, Rokhlin, Tygert, Woolfe (2006).
  - The SRFT (under the name *fast Johnson-Lindenstrauss transform*) was suggested by Nir Ailon and Bernard Chazelle (2006) in a related context.
  - Dissertation by Edo Liberty.
  - Related recent work by Sarlós (on randomized regression).
  - Overdetermined linear least-squares regression: Rokhlin and Tygert (2008).

Input: An $m \times n$ matrix **A** and a target rank k.	
Output: Rank-k factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.	
(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Cost of steps 2 and 4: Application of **A** and $\mathbf{A}^*$ to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

Case 2: **A** is presented as an array of real numbers in slow memory — on "disk".

In this case, standard methods such as Golub-Businger become prohibitively slow due to the random access requirements on the data.

However, the method described above works just fine. (Recall: Two passes only!)

Limitation 1: Matrices of size $m \times k$ and $k \times n$ must fit in RAM.

Limitation 2: For matrices whose singular values decay slowly (as is typical in the data-analysis environment), the method above is typically not accurate enough.
Higher accuracy can be bought at modest cost — we will return to this point.

Code: Webpage of Yoel Shkolnisky.

More code:


Firefox
Stochastic Singular Value Deco...
https://cwiki.apache.org/confluence/display/MAHOUT/Stochastic+Singular+Vi
Google
mahout
Dashboard › Apache Mahout › ... › Algorithms › Stochastic Singular Value Decomposition
Added by Dmitriy Lyubimov, last edited by Dmitriy Lyubimov on May 07, 2012 (view change)
Stochastic Singular Value Decomposition
Stochastic SVD method in Mahout produces reduced rank Singular Value Decomposition output in its strict mathematical definition: A=USV'.
The benefits over other methods are:
reduced flops required compared to Krylov subspace methods
In map-reduce world, a fixed number of MR iterations required regardless of rank requested
Tweak precision/speed balance with options.
A is a Distributed Row Matrix where rows may be identified by any Writable (such as a document path). As such, it would work directly on the output of seq2sparse.
As of 0.7 trunk, includes PCA and dimensionality reduction workflow (EXPERIMENTAL! Feedback on performance/other PCA related issues/ blogs is greatly appreciated.)
map-reduce characteristics:
SSVD uses at most 3 MR sequential steps (map-only + map-reduce + 2 optional parallel map-reduce jobs) to produce reduced rank approximation of U, V and S matrices. Additionally, two more map-reduce steps are added for each power iteration step if requested.
Potential drawbacks:
potentially less precise (but adding even one power iteration seems to fix that quite a bit).
Documentation
Overview and Usage

Input: An $m \times n$ matrix **A** and a target rank k.

Output: Rank-k factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Cost of steps 2 and 4: Application of **A** and $\mathbf{A}^*$ to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

Case 3: **A** and $\mathbf{A}^*$ admit fast matrix-vector multiplication.

In this case, the "standard" method is some variation of Krylov methods such as Lanczos (or Arnoldi for non-symmetric matrices) whose cost T_{Krylov} satisfy:

$$T_{\text{Krylov}} \sim k\, T_{\text{mat-vec-mult}} + \underbrace{O(k^2\,(m+n))}_{\text{With full "reorthogonalization".}} .$$

The asymptotic cost of the randomized scheme is the same; its advantage is again in how the data is accessed — the k matrix-vector multiplies can be executed in parallel.

The method above is in important environments less accurate than Arnoldi,
but this can be fixed while compromising only slightly on the pass-efficiency.

Theory

Input: An $m \times n$ matrix $\mathbf{A}$ and a target rank k.

Output: Rank-k factors $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What is the error $e_k = ||\mathbf{A} - \mathbf{U\Sigma V}^*||$? (Recall $e_k = ||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}||$.)

Input: An $m \times n$ matrix **A** and a target rank k.

Output: Rank-k factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What is the error $e_k = ||\mathbf{A} - \mathbf{U\Sigma V}^*||$? (Recall $e_k = ||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}||$.)

Eckart-Young theorem: e_k is bounded from below by the $(k+1)$'th singular value σ_{k+1} of **A**.

Question: Is e_k close to σ_{k+1}?

Input: An $m \times n$ matrix **A** and a target rank k.

Output: Rank-k factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

(1) Draw an $n \times k$ random matrix $\mathbf{\Omega}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What is the error $e_k = ||\mathbf{A} - \mathbf{U\Sigma V}^*||$? (Recall $e_k = ||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}||$.)

Eckart-Young theorem: e_k is bounded from below by the $(k+1)$'th singular value σ_{k+1} of **A**.

Question: Is e_k close to σ_{k+1}?

Answer: Lamentably, no. The expectation of $\frac{e_k}{\sigma_{k+1}}$ is large, and has very large variance.

Input: An $m \times n$ matrix **A** and a target rank k.	
Output: Rank-k factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.	
(1) Draw an $n \times k$ random matrix **Ω**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Question: What is the error $e_k = ||\mathbf{A} - \mathbf{U\Sigma V}^*||$? (Recall $e_k = ||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}||$.)

Eckart-Young theorem: e_k is bounded from below by the $(k+1)$'th singular value σ_{k+1} of **A**.

Question: Is e_k close to σ_{k+1}?

Answer: Lamentably, no. The expectation of $\frac{e_k}{\sigma_{k+1}}$ is large, and has very large variance.

Remedy: Over-sample slightly. Compute $k+p$ samples from the range of **A**.
It turns out that $p = 5$ or 10 is often sufficient. $p = k$ is almost always more than enough.

Input: An $m \times n$ matrix **A**, a target rank k, *and an over-sampling parameter p (say $p = 5$).*	
Output: Rank-$(k+p)$ factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **Ω**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Bound on the expectation of the error for Gaussian test matrices

Let **A** denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

Let k denote a target rank and let p denote an over-sampling parameter.

Let **Ω** denote an $n \times (k+p)$ Gaussian matrix.

Let **Q** denote the $m \times (k+p)$ matrix $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$.

If $p \geq 2$, then

$$\mathbb{E}||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}||_{\text{Frob}} \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2},$$

and

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

Ref: Halko, Martinsson, Tropp, 2009 & 2011

Large deviation bound for the error for Gaussian test matrices

Let **A** denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

Let k denote a target rank and let p denote an over-sampling parameter.

Let **Ω** denote an $n \times (k+p)$ Gaussian matrix.

Let **Q** denote the $m \times (k+p)$ matrix $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$.

If $p \geq 4$, and u and t are such that $u \geq 1$ and $t \geq 1$, then

$$||\mathbf{A} - \mathbf{QQ}^* \mathbf{A}|| \leq \left(1 + t \sqrt{\frac{3k}{p+1}} + u\, t\, \frac{e\sqrt{k+p}}{p+1} \right) \sigma_{k+1} + \frac{t\, e \sqrt{k+p}}{p+1} \left(\sum_{j>k} \sigma_j^2 \right)^{1/2}$$

except with probability at most $2\, t^{-p} + e^{-u^2/2}$.

Ref: Halko, Martinsson, Tropp, 2009 & 2011; Martinsson, Rokhlin, Tygert (2006)

---

u and t parameterize "bad" events — large u, t is bad, but unlikely.

Certain choices of t and u lead to simpler results. For instance,

$$||\mathbf{A} - \mathbf{QQ}^* \mathbf{A}|| \leq \left(1 + 16 \sqrt{1 + \frac{k}{p+1}} \right) \sigma_{k+1} + 8 \frac{\sqrt{k+p}}{p+1} \left(\sum_{j>k} \sigma_j^2 \right)^{1/2},$$

except with probability at most $3\, e^{-p}$.

Large deviation bound for the error for Gaussian test matrices

Let **A** denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

Let k denote a target rank and let p denote an over-sampling parameter.

Let **Ω** denote an $n \times (k+p)$ Gaussian matrix.

Let **Q** denote the $m \times (k+p)$ matrix $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$.

If $p \geq 4$, and u and t are such that $u \geq 1$ and $t \geq 1$, then

$$||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}|| \leq \left(1 + t\sqrt{\frac{3k}{p+1}} + u\,t\,\frac{e\sqrt{k+p}}{p+1}\right)\sigma_{k+1} + \frac{t\,e\sqrt{k+p}}{p+1}\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$$

except with probability at most $2\,t^{-p} + e^{-u^2/2}$.

Ref: Halko, Martinsson, Tropp, 2009 & 2011; Martinsson, Rokhlin, Tygert (2006)

---

u and t parameterize "bad" events — large u, t is bad, but unlikely.

Certain choices of t and u lead to simpler results. For instance,

$$||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}|| \leq \left(1 + 6\sqrt{(k+p)\cdot p\log p}\right)\sigma_{k+1} + 3\sqrt{k+p}\left(\sum_{j>k}\sigma_j^2\right)^{1/2},$$

except with probability at most $3\,p^{-p}$.

Proofs — Overview:

Let **A** denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

Let k denote a target rank and let p denote an over-sampling parameter. Set $\ell = k + p$.

Let $\mathbf{\Omega}$ denote an $n \times \ell$ "test matrix", and let **Q** denote the $m \times \ell$ matrix $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$.

We seek to bound the error $e_k = e_k(\mathbf{A}, \mathbf{\Omega}) = ||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}||$, which is a random variable.

1. Make no assumption on $\mathbf{\Omega}$. Construct a deterministic bound of the form

$$||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}|| \leq \cdots \mathbf{A} \cdots \mathbf{\Omega} \cdots$$

2. Assume that $\mathbf{\Omega}$ is drawn from a normal Gaussian distribution.
   Take expectations of the deterministic bound to attain a bound of the form

$$\mathbb{E}\big[||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}||\big] \leq \cdots \mathbf{A} \cdots$$

3. Assume that $\mathbf{\Omega}$ is drawn from a normal Gaussian distribution.
   Take expectations of the deterministic bound conditioned on "bad behavior" in $\mathbf{\Omega}$ to get

$$||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}|| \leq \cdots \mathbf{A} \cdots$$

   that hold with probability at least $\cdots$.

Part 1 (out of 3) — deterministic bound:

Let **A** denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

Let k denote a target rank and let p denote an over-sampling parameter. Set $\ell = k + p$.

Let $\mathbf{\Omega}$ denote an $n \times \ell$ "test matrix", and let **Q** denote the $m \times \ell$ matrix $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$.

Partition the SVD of **A** as follows:

$$\mathbf{A} = \mathbf{U} \begin{array}{c} \\ k \\ n-k \end{array} \begin{array}{c} \begin{array}{cc} k & n-k \end{array} \\ \begin{bmatrix} \mathbf{\Sigma}_1 & \\ & \mathbf{\Sigma}_2 \end{bmatrix} \end{array} \begin{array}{c} \\ \begin{bmatrix} \mathbf{V}_1^* \\ \mathbf{V}_2^* \end{bmatrix} \end{array} \begin{array}{c} \\ k \\ n-k \end{array}$$

Define $\mathbf{\Omega}_1$ and $\mathbf{\Omega}_2$ via

$$\mathbf{\Omega}_1 = \mathbf{V}_1^* \, \mathbf{\Omega} \quad \text{and} \quad \mathbf{\Omega}_2 = \mathbf{V}_2^* \, \mathbf{\Omega}.$$

Theorem: [HMT2009,HMT2011] Assuming that $\mathbf{\Omega}_1$ is not singular, it holds that

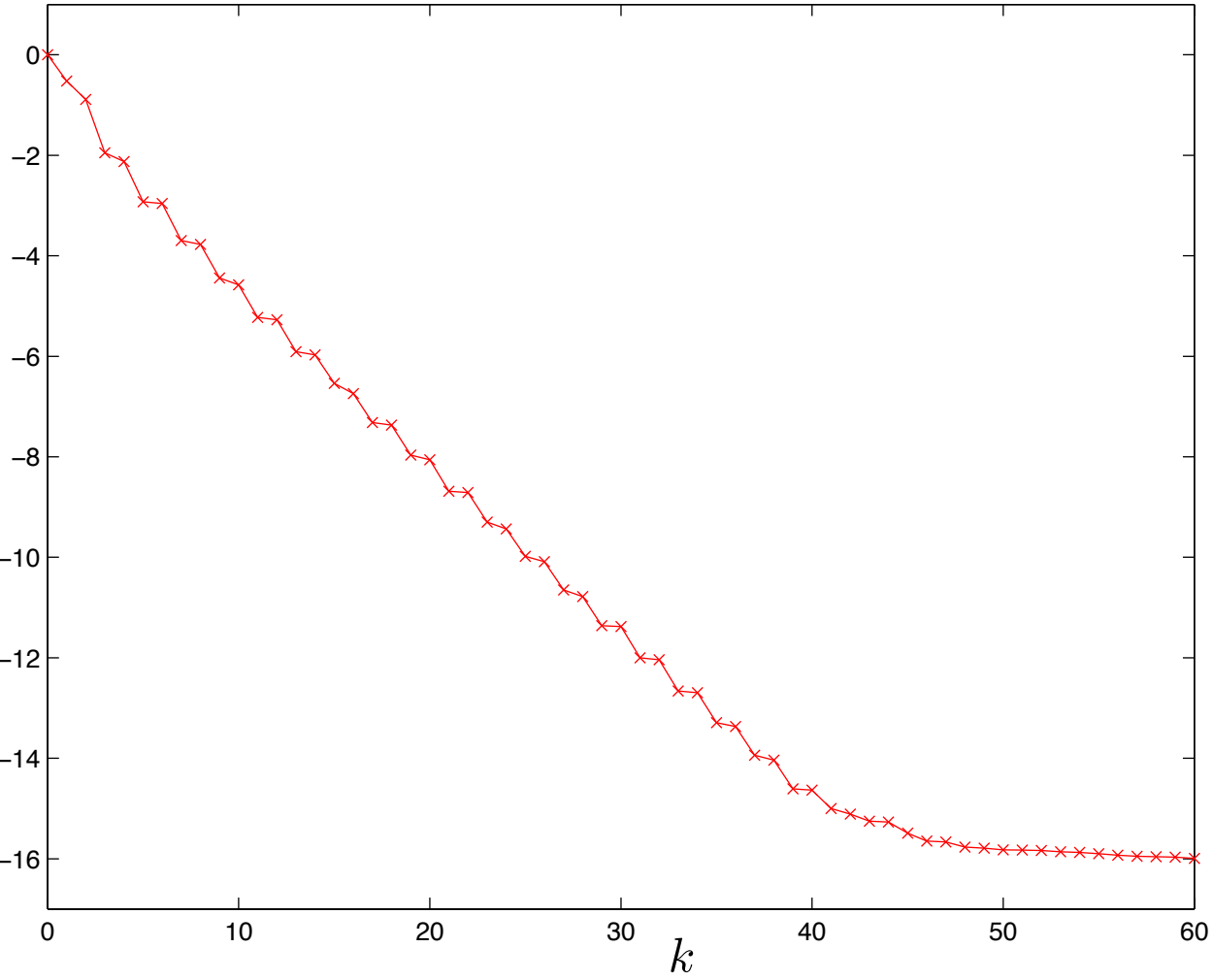
$$|||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}|||^2 \leq \underbrace{|||\mathbf{\Sigma}_2|||^2}_{\text{theoretically minimal error}} + |||\mathbf{\Sigma}_2\mathbf{\Omega}_2\mathbf{\Omega}_1^\dagger|||^2.$$

Here, $|||\cdot|||$ represents either ℓ^2-operator norm, or the Frobenius norm.

Note: A similar (but weaker) result appears in Boutsidis, Mahoney, Drineas (2009).

Example 1:

We consider a $1\,000 \times 1\,000$ matrix **A** whose singular values are shown below:

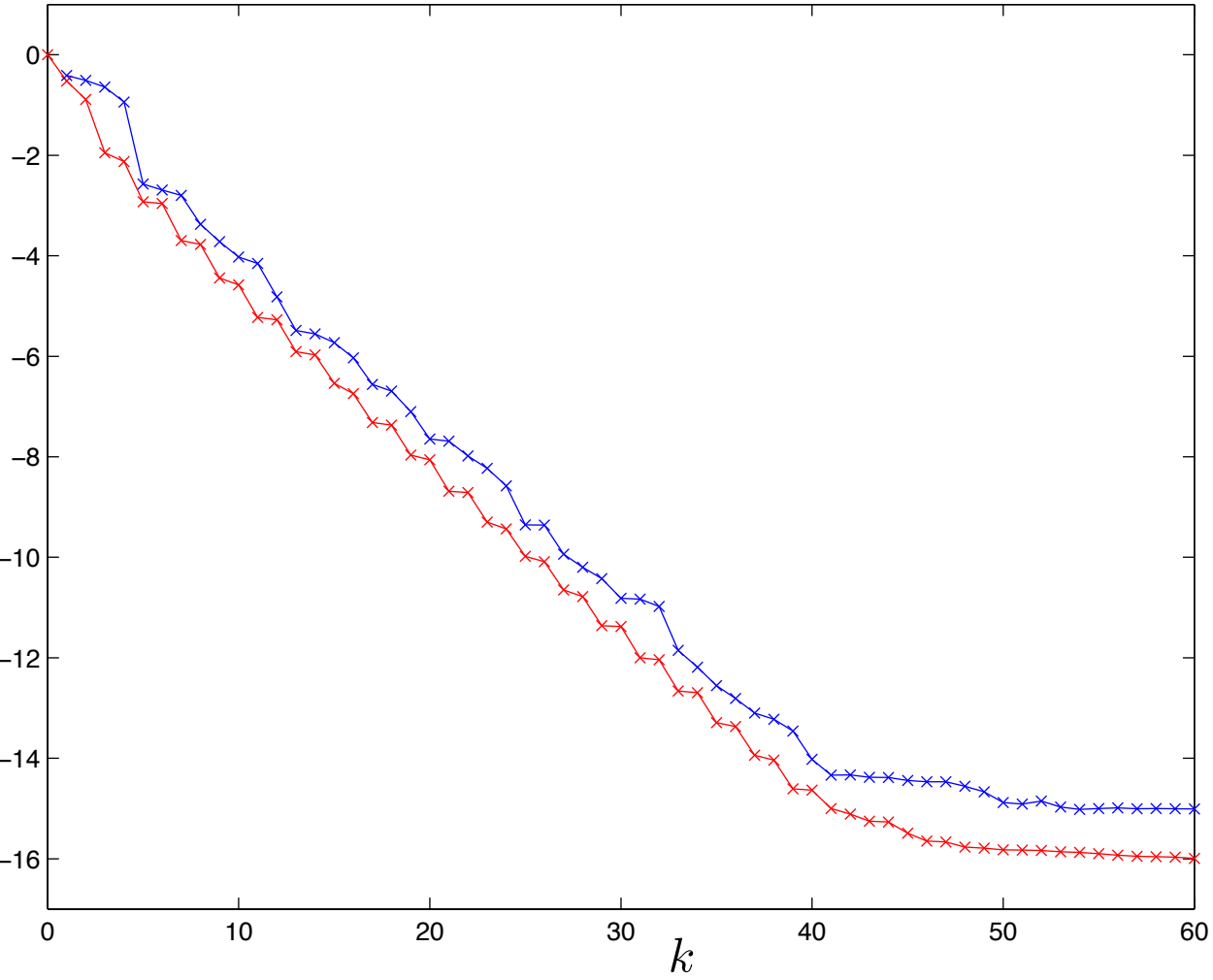


The red line indicates the singular values σ_{k+1} of **A**. These indicate the theoretically minimal approximation error.

A is a discrete approximation of a certain compact integral operator. Curiously, the nature of **A** is in a strong sense irrelevant: the error distribution depends only on $(\sigma_j)_{j=1}^{\min(m,n)}$.

Example 1:

We consider a $1\,000 \times 1\,000$ matrix **A** whose singular values are shown below:



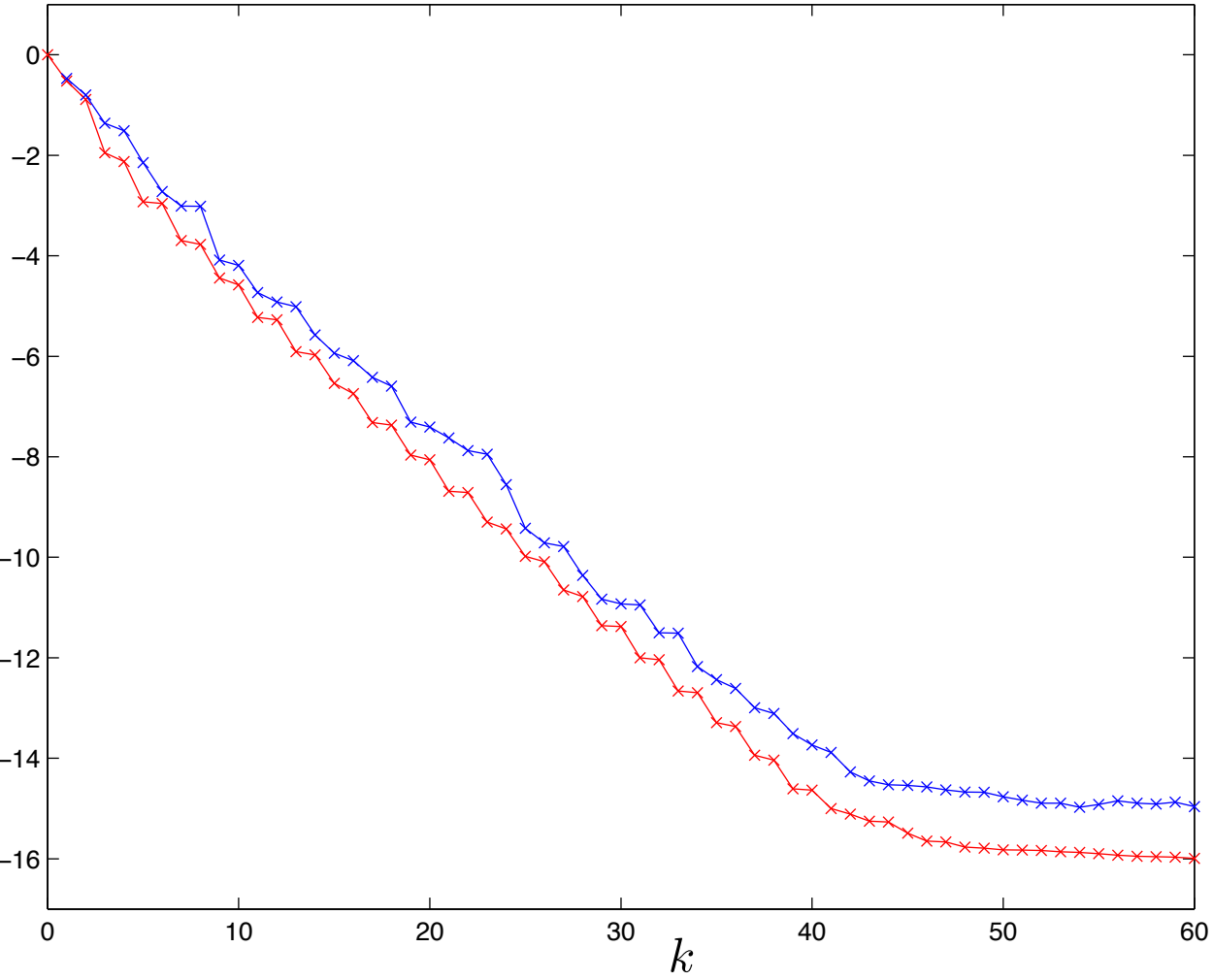
The red line indicates the singular values σ_{k+1} of **A**. These indicate the theoretically minimal approximation error.

The blue line indicates the actual errors e_k incurred by one instantiation of the proposed method.

A is a discrete approximation of a certain compact integral operator. Curiously, the nature of **A** is in a strong sense irrelevant: the error distribution depends only on $(\sigma_j)_{j=1}^{\min(m,n)}$.

Example 1:

We consider a $1\,000 \times 1\,000$ matrix **A** whose singular values are shown below:



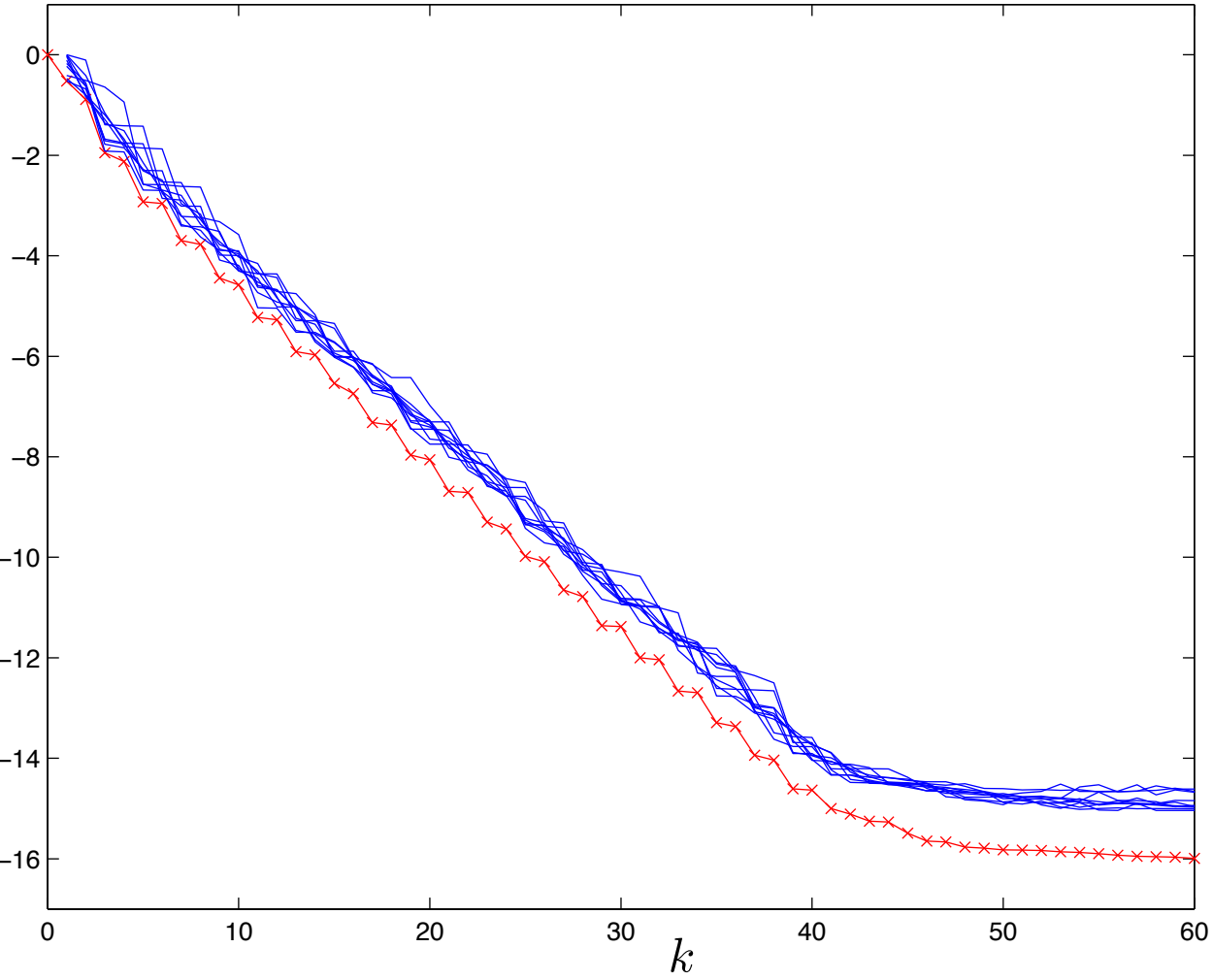
The red line indicates the singular values σ_{k+1} of **A**. These indicate the theoretically minimal approximation error.

The blue line indicates the actual errors e_k incurred by a different instantiation of the proposed method.

A is a discrete approximation of a certain compact integral operator. Curiously, the nature of **A** is in a strong sense irrelevant: the error distribution depends only on $(\sigma_j)_{j=1}^{\min(m,n)}$.

Example 1:

We consider a $1\,000 \times 1\,000$ matrix **A** whose singular values are shown below:



The red line indicates the singular values σ_{k+1} of **A**. These indicate the theoretically minimal approximation error.

The blue lines indicate the actual errors e_k incurred by 10 different instantiations of the proposed method.

A is a discrete approximation of a certain compact integral operator. Curiously, the nature of **A** is in a strong sense irrelevant: the error distribution depends only on $(\sigma_j)_{j=1}^{\min(m,n)}$.

Power method for improving accuracy:

The error depends on how quickly the singular values decay. Recall that

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

The faster the singular values decay — the stronger the relative weight of the dominant modes in the samples.

Idea: The matrix $(\mathbf{A}\,\mathbf{A}^*)^q\,\mathbf{A}$ has the same left singular vectors as $\mathbf{A}$, and singular values

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

Much faster decay — so let us use the sample matrix

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

instead of

$$\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega}.$$

References: Paper by Rokhlin, Szlam, Tygert (2008). Suggestions by Ming Gu. Also similar to “block power method,” “block Lanczos,” “subspace iteration.”

Input: An $m \times n$ matrix **A**, a target rank ℓ, and a small integer q.	
Output: Rank-ℓ factors **U**, **Σ**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.	
(1) Draw an $n \times \ell$ random matrix **Ω**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $n \times \ell$ sample matrix $\mathbf{Y} = (\mathbf{A}\,\mathbf{A}^*)^q\,\mathbf{A\Omega}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma V}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

- Detailed (and, we believe, close to sharp) error bounds have been proven.
  For instance, the expectation of the error satisfies:

$$\mathbb{E}\left[\,||\mathbf{A} - \mathbf{A}_{k+p}^{\text{computed}}||\,\right] \leq \left(1 + 4\sqrt{\frac{2\,\min(m,n)}{k-1}}\right)^{1/(2q+1)} \sigma_{k+1}(\mathbf{A}). \tag{15}$$

  Reference: Halko, Martinsson, Tropp (2011).

- The improved accuracy from the modified scheme comes at a cost;
  $2q+1$ passes over the matrix are required instead of 1.
  However, q can often be chosen quite small in practice, $q = 2$ or $q = 3$, say.

- The bound (15) assumes exact arithmetic.
  To handle round-off errors, variations of subspace iterations can be used.
  These are entirely numerically stable and achieve the same error bound.

A numerically stable version of the "power method":

Input: An $m \times n$ matrix $\mathbf{A}$, a target rank ℓ, and a small integer q.
Output: Rank-ℓ factors $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

Draw an $n \times \ell$ Gaussian random matrix $\mathbf{\Omega}$.
Set $\mathbf{Q} = \mathtt{orth}(\mathbf{A\Omega})$
for $i = 1, 2, \ldots, q$
    $\mathbf{W} = \mathtt{orth}(\mathbf{A}^*\mathbf{Q})$
    $\mathbf{Q} = \mathtt{orth}(\mathbf{AW})$
end for
$\mathbf{B} = \mathbf{Q}^*\mathbf{A}$
$[\hat{\mathbf{U}}, \mathbf{\Sigma}, \mathbf{V}] = \mathtt{svd}(\mathbf{B})$
$\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

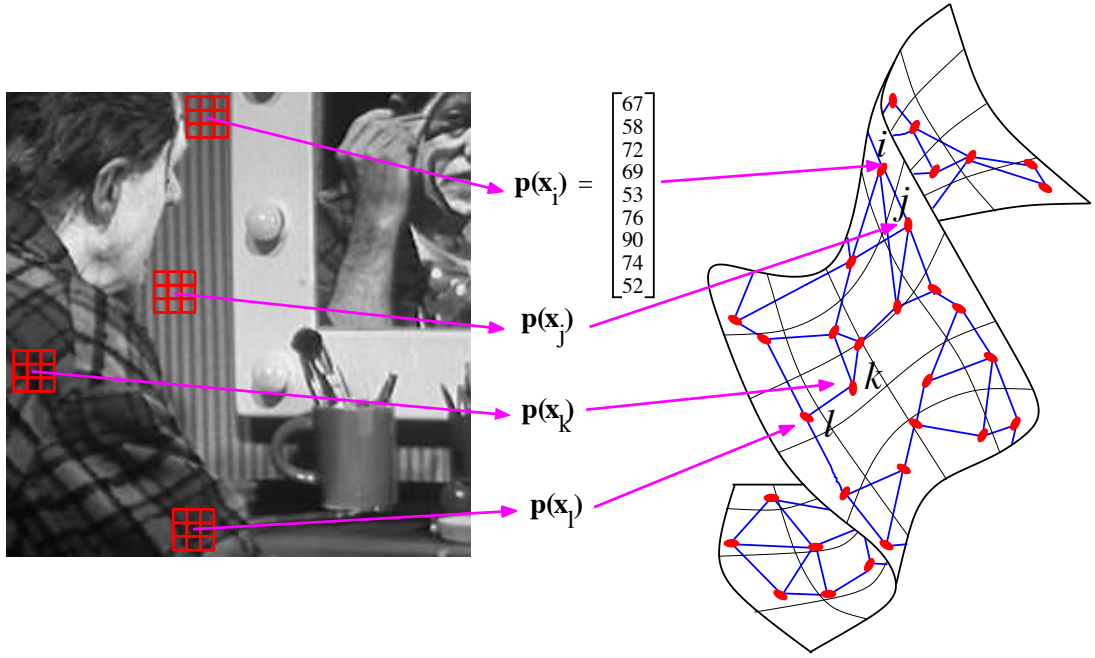
Note: Algebraically, the method with orthogonalizations is identical to the "original" method where $\mathbf{Q} = \mathtt{orth}((\mathbf{AA}^*)^q\mathbf{A\Omega})$.

Note: This is a classic subspace iteration.
The novelty is the error analysis, and the finding that using a very small q is often fine.
(In fact, our analysis allows q to be zero...)

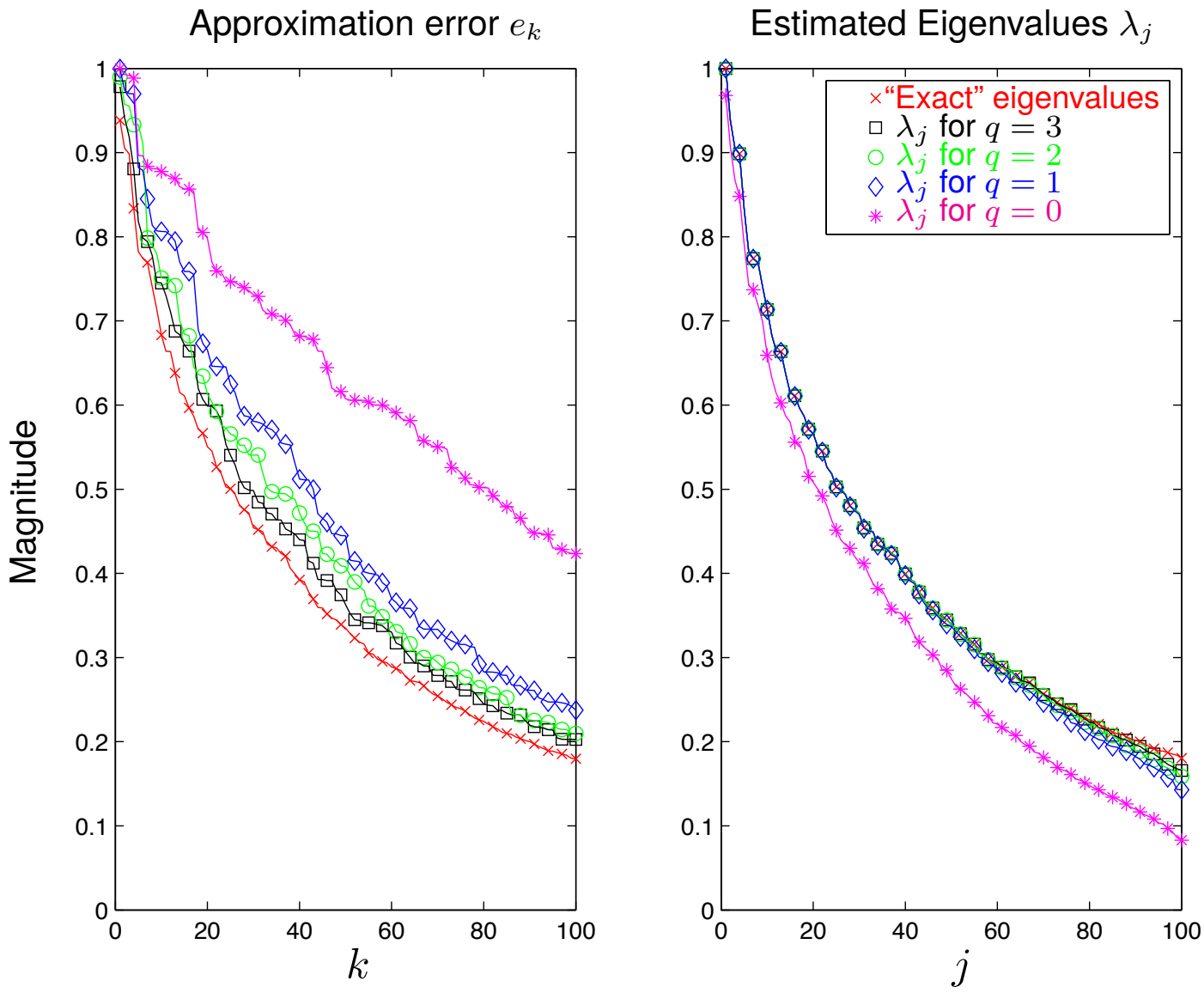
Example 2 (revisited):

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

To be precise, **A** is a graph Laplacian on the manifold of 3×3 patches.



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



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

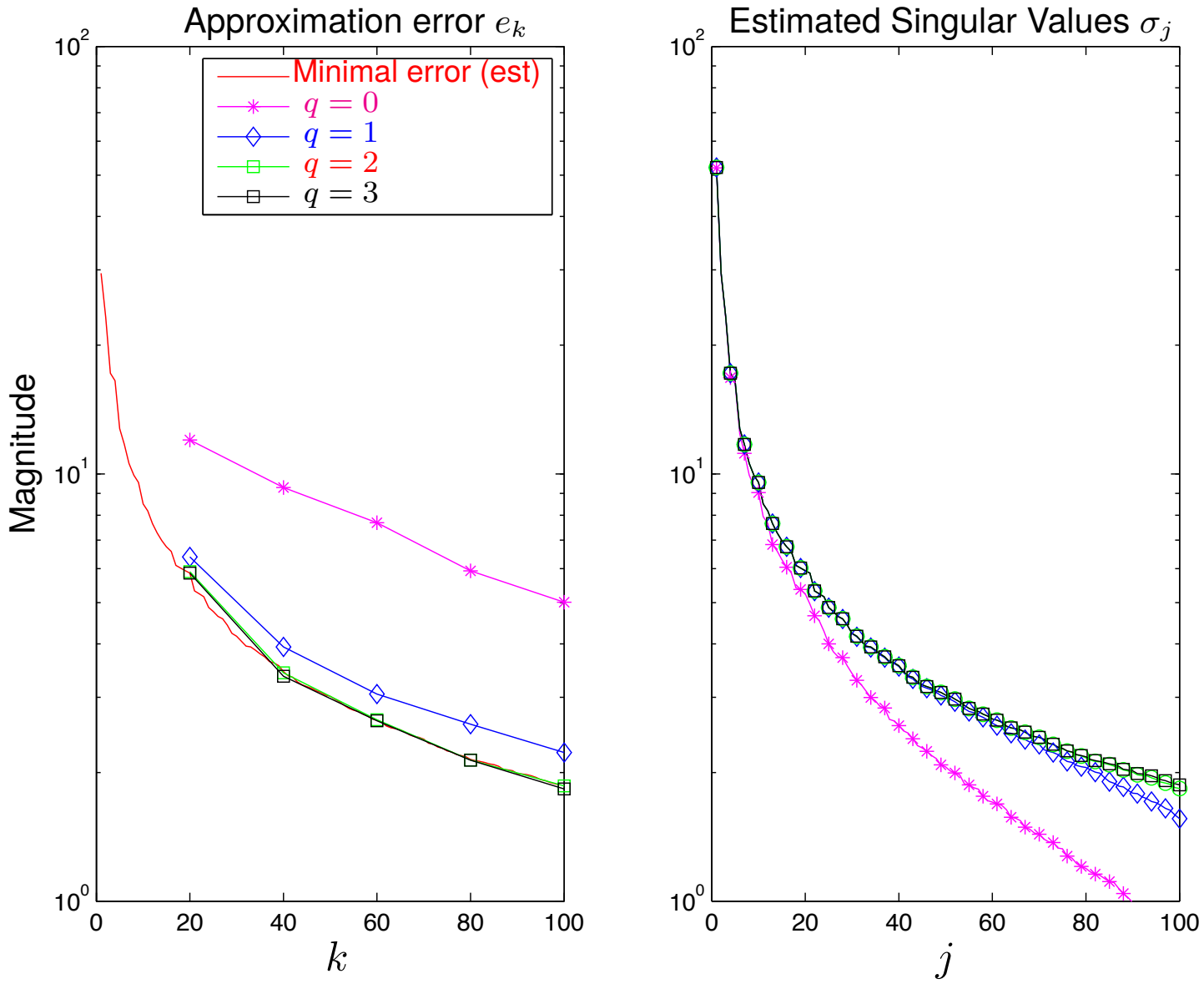
Example 3 (revisited): "Eigenfaces"

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

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

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

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



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

Example of application: Nearest neighbor search in $\mathbb{R}^D$

Peter Jones, Andrei Osipov, Vladimir Rokhlin

Suppose you are given N points $\{\boldsymbol{x}_j\}_{j=1}^N$ in $\mathbb{R}^D$. The coordinate matrix is

$$\mathbf{X} = [\boldsymbol{x}_1\ \boldsymbol{x}_2\ \cdots \boldsymbol{x}_N] \in \mathbb{R}^{D\times N}.$$

How do you find the k nearest neighbors for every point?

If $D = 2$ there are several options; you can, e.g, sort the points into a quad-tree. Standard techniques get very expensive as D grows.

Simple idea: Use a random map to project onto low-dimensional space. This "sort of" preserves distances. Execute a fast search there.

Improved idea: The output from a single random projection is unreliable. But, you can repeat the experiment several times, use these to generate a list of *candidates* for the nearest neighbors, and then compute exact distances to find the k closest among the candidates.

Final remarks:

- For large scale SVD/PCA of dense matrices, these algorithms are highly recommended; they compare favorably to existing methods in almost every regard. Free software can be downloaded $\rightarrow$ google *Mark Tygert*.

- The approximation error is a random variable, but its distribution is very narrowly concentrated. Rigorous error bounds that are satisfied with probability $1 - \eta$ where η is a user set "failure probability" (*e.g.* $\eta = 10^{-10}$ or 10^{-20}).

- This talk mentioned *error estimators* only briefly, but they are important.
  Can operate independently of the algorithm for improved robustness.
  Typically cheap and easy to implement. Used to determine the actual rank.

- The theory can be hard, but *experimentation is easy!*
  Concentration of measure makes the algorithms behave as if deterministic.

- Randomized methods for computing "FMM"-style (HSS, $\mathcal{H}$-matrix, ...) representations of matrices have been developed — [Martinsson 2008 & 2011], [Lin, Lu, Ying 2011].

- Survey paper: N. Halko, P.G. Martinsson, J. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.* SIAM Review, **53**(2), 2011, pp. 217–288.
  Slides: Under "Talks" tab on my webpage (google "Gunnar Martinsson").

We note that the power method can be viewed as a hybrid between the "basic" randomized method, and a Krylov subspace method:

Krylov method: Restrict $\mathbf{A}$ to the linear space

$$\mathcal{V}_q(\boldsymbol{\omega}) = \text{Span}(\mathbf{A}\boldsymbol{\omega},\ \mathbf{A}^2\boldsymbol{\omega},\ \ldots,\ \mathbf{A}^q\boldsymbol{\omega}).$$

"Basic" randomized method: Restrict $\mathbf{A}$ to the linear space

$$\text{Span}(\mathbf{A}\boldsymbol{\omega}_1,\ \mathbf{A}\boldsymbol{\omega}_2,\ \ldots,\ \mathbf{A}\boldsymbol{\omega}_\ell) = \mathcal{V}_1(\boldsymbol{\omega}_1) \times \mathcal{V}_1(\boldsymbol{\omega}_2) \times \cdots \times \mathcal{V}_1(\boldsymbol{\omega}_\ell).$$

"Power" method: Restrict $\mathbf{A}$ to the linear space

$$\text{Span}(\mathbf{A}^q\boldsymbol{\omega}_1,\ \mathbf{A}^q\boldsymbol{\omega}_2,\ \ldots,\ \mathbf{A}^q\boldsymbol{\omega}_\ell).$$

Modified "power" method: Restrict $\mathbf{A}$ to the linear space

$$\mathcal{V}_q(\boldsymbol{\omega}_1) \times \mathcal{V}_q(\boldsymbol{\omega}_2) \times \cdots \times \mathcal{V}_q(\boldsymbol{\omega}_\ell).$$

This could be a promising area for further work.

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*

$$k \geq 4 \left(\frac{\varepsilon^2}{2} - \frac{\varepsilon^3}{3} \right)^{-1} \log(n). \tag{16}$$

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

$$(1 - \varepsilon) \, ||u - v||^2 \leq ||f(u) - f(v)|| \leq (1 + \varepsilon) \, ||u - v||^2, \qquad \forall \, u, \, v \in V. \tag{17}$$

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, (16), this map satisfies (17) 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, Tao, 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 in the Johnson-Lindenstrauss theorem:

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

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 stable fast 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.