Randomized algorithms for linear algebraic computations

Per-Gunnar Martinsson
Dept. of Mathematics & Oden Institute for Computational Sciences and Engineering
University of Texas at Austin

Collaborators: Robert van de Geijn, Francisco Igual, Yuji Nakatsukasa, Gregorio Quintana-Ortí, Vladimir Rokhlin, Joel Tropp, Mark Tygert.

Former and current students and postdocs: Ke Chen, Yijun Dong, Abinand Gopal, Nathan Halko, Nathan Heavner, James Levitt, Sergey Voronin, Heather Wilber, Bowei Wu, Anna Yesypenko.

Slides: `http://users.oden.utexas.edu/~pgm/main_talks.html`

Research support by:

Wednesday: Randomized algorithms for linear algebraic computations

1. **Randomized low rank approximation:** "Randomized singular value decomposition" or "RSVD". Relatively well established material.
2. **Variations of algorithms for low rank approximation:** Single pass and streaming algorithms. Structured random embeddings. Matrix approximation via sampling.
3. **Samples of current research directions (time permitting):** Linear solvers. Least squares problems. Block Krylov methods. Rank structured matrices.

Friday: Randomized embeddings — theory and applications

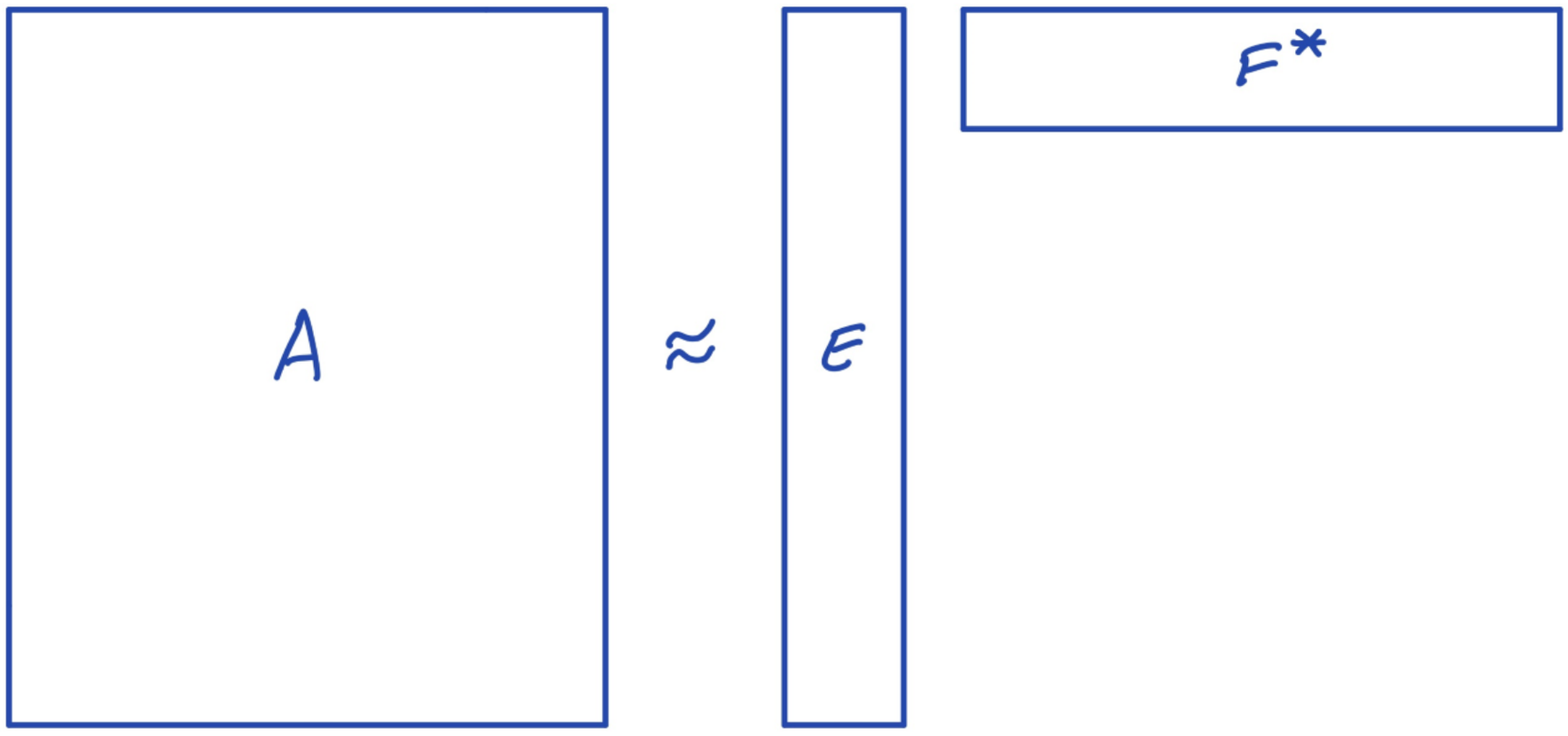
4. **Randomized embeddings:** Reducing the effective dimension of point sets. Connections to Johnson-Lindenstrauss theory. Norm estimation.
5. **Analysis of the RSVD:** Outline of probabilistic error analysis for the RSVD. The relative merits of different classes of randomized embeddings.
6. **The column/row selection problem:** Interpolatory and CUR decompositions. Pivoting in QR and LU factorizations.

Low rank approximation — problem formulation:

Let **A** be a given $m \times n$ matrix, and let k be an integer such that $1 \leq k \ll n \leq m$.

We seek to compute approximate factors **E** and **F** such that

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{E} & \mathbf{F}^*. \\ m \times n & & m \times k & k \times n \end{array}$$



Low rank approximation — problem formulation:

Let **A** be a given $m \times n$ matrix, and let k be an integer such that $1 \leq k \ll n \leq m$.

We seek to compute approximate factors **E** and **F** such that

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{E}} \underset{k \times n}{\mathbf{F}^*}.$$

Why?

- Fitting a hyperplane to a given set of points. Or fitting a multivariate normal distribution to measurements ("principal component analysis").
- Model reduction in scientific computing.
- Spectral algorithms in data analysis.
- "Fast" algorithms of various types: Fast Multipole Methods, generalizations of the Fast Fourier Transform, fast direct solvers, etc.
- Many, many, many more.

Observe that from **E** and **F** you can compute approximate singular vectors, find dominant eigenvectors (when **A** is normal at least), find spanning rows/columns, etc.

We seek only to control the residual error $\|\mathbf{A} - \mathbf{E}\mathbf{F}^*\|$.

Low rank approximation — problem formulation:

Let **A** be a given $m \times n$ matrix, and let k be an integer such that $1 \leq k \ll n \leq m$.

We seek to compute approximate factors **E** and **F** such that

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{E} & & \mathbf{F}^*. \\ m \times n & & m \times k & & k \times n \end{array}$$

Existing methods for this task are well established. Textbook methods include:

1. Compute the full singular value decomposition of **A**, and then truncate:
   — Resulting approximation is in many regards ”optimal” — best possible fit.
   — Expensive! Cost is $O(mn^2)$. Good for small n, or “expensive” data.
2. Krylov methods:
   — Standard technique for large sparse matrices.
   — Interacts with **A** only through its action on vectors. Cost $\sim k \times T_{\text{matvec}}$.
   — Theoretically optimal in important regards.
3. Execute Gram-Schmidt on the columns (or rows) of **A** — “column pivoted QR”:
   — Simple and practical for medium size dense matrices.
   — Not entirely optimal, but often good enough.
   — Cost is $O(mnk)$ since you can stop after k steps.

These methods work great! But room for improvement in important environments.

Randomized SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^* \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m, n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`

A.2 Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

The objective of Stage A is to compute an ON-basis that approximately spans the column space of $\mathbf{A}$. The matrix $\mathbf{Q}$ holds these basis vectors and $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$.

Randomized SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^* \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m, n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`

A.2 Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

The objective of Stage A is to compute an ON-basis that approximately spans the column space of $\mathbf{A}$. The matrix $\mathbf{Q}$ holds these basis vectors and $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$.

Stage B is exact: $\|\mathbf{A} - \mathbf{Q}\underbrace{\mathbf{Q}^*\mathbf{A}}_{=\mathbf{B}}\| = \|\mathbf{A} - \mathbf{Q}\underbrace{\mathbf{B}}_{=\hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*}\| = \|\mathbf{A} - \underbrace{\mathbf{Q}\hat{\mathbf{U}}}_{=\mathbf{U}}\mathbf{D}\mathbf{V}^*\| = \|\mathbf{A} - \mathbf{UDV}^*\|$.

Randomized SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^* \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m, n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`

A.2 Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

We claim that the columns of $\mathbf{Y}$ form a good approximate basis for $\text{ran}(\mathbf{A})$.

Observe that $\text{ran}(\mathbf{Y}) \subseteq \text{ran}(\mathbf{A})$ automatically.

Loss of accuracy can happen if $\text{ran}(\mathbf{Y})$ does not capture important directions.

To avoid this, we draw p extra samples, for, say, $p = 5$ or $p = 10$.

Randomized SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^* \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m, n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{\Omega}$. `Omega = randn(n,k)`

A.2 Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{A\Omega}$. `Y = A * Omega`

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

Important: You only need to ensure that you do not undersample.

Over-sampling is unproblematic, since excess data gets "filtered out" in Stage B.

Randomized SVD:

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

Randomized SVD:

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

- It is simple to adapt the scheme to the situation where the *tolerance is given,* and the rank has to be determined adaptively.

- Observe how simple the interaction with $\mathbf{A}$ is.

  Two matrix-matrix products only.

  This often leads to very high practical execution speed on modern hardware.

- Accuracy of the basic scheme is good when the singular values decay reasonably fast. When they do not, the scheme can be combined with Krylov-type ideas:

  Taking one or two steps of subspace iteration vastly improves the accuracy.

  For instance, use the sampling matrix $\mathbf{Y} = \mathbf{AA}^*\mathbf{A\Omega}$ instead of $\mathbf{Y} = \mathbf{A\Omega}$.

Randomized SVD:

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

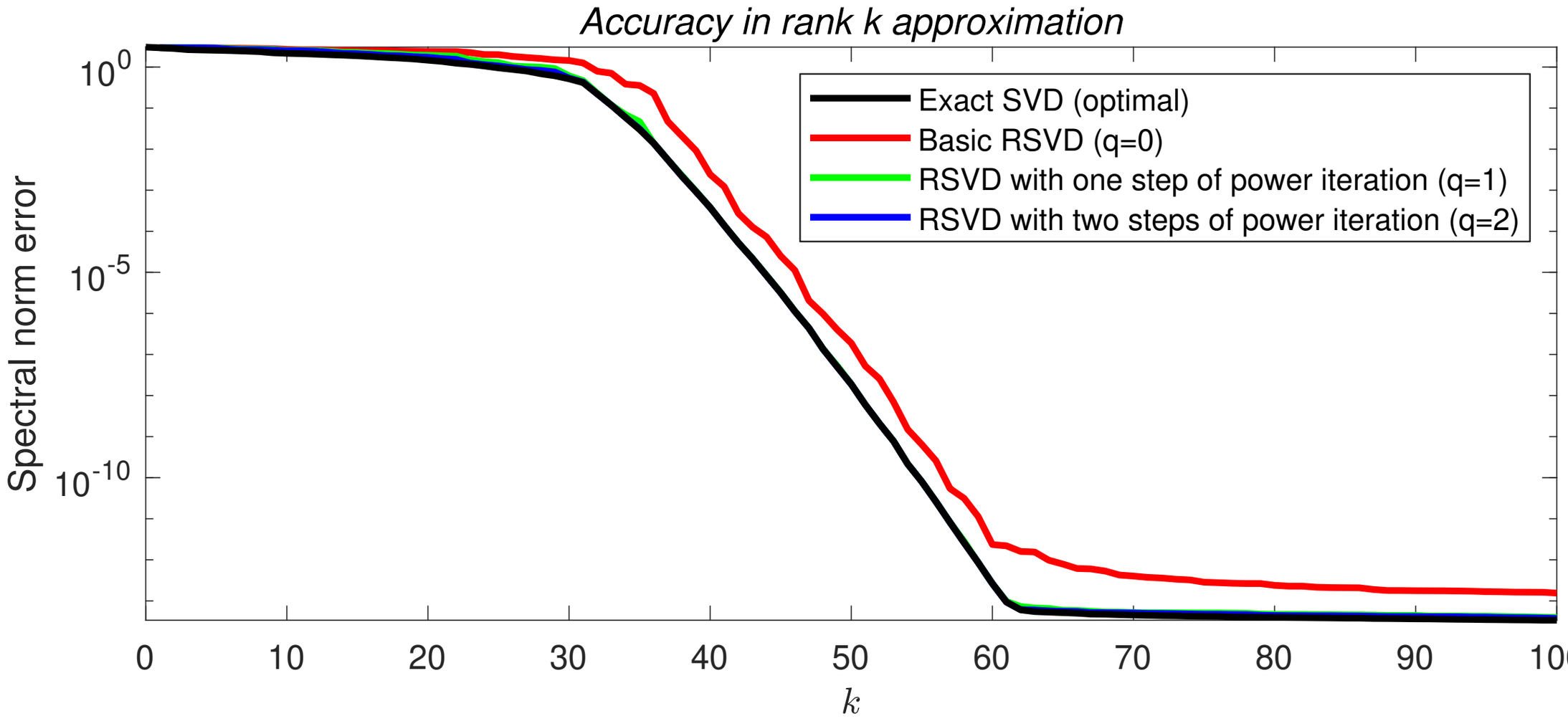
Let us next investigate the accuracy of the method.

To illustrate the errors, we first set $p = 0$ (no over-sampling), and then define

$$e_k = \|\mathbf{A} - \mathbf{UDV}^*\| = \|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|.$$

Eckart-Young theorem: $e_k \geq \sigma_{k+1}$, where σ_{k+1} is the $(k+1)$'th singular value of $\mathbf{A}$.

Randomized SVD:



The plot shows the errors from the randomized SVD. To be precise, we plot

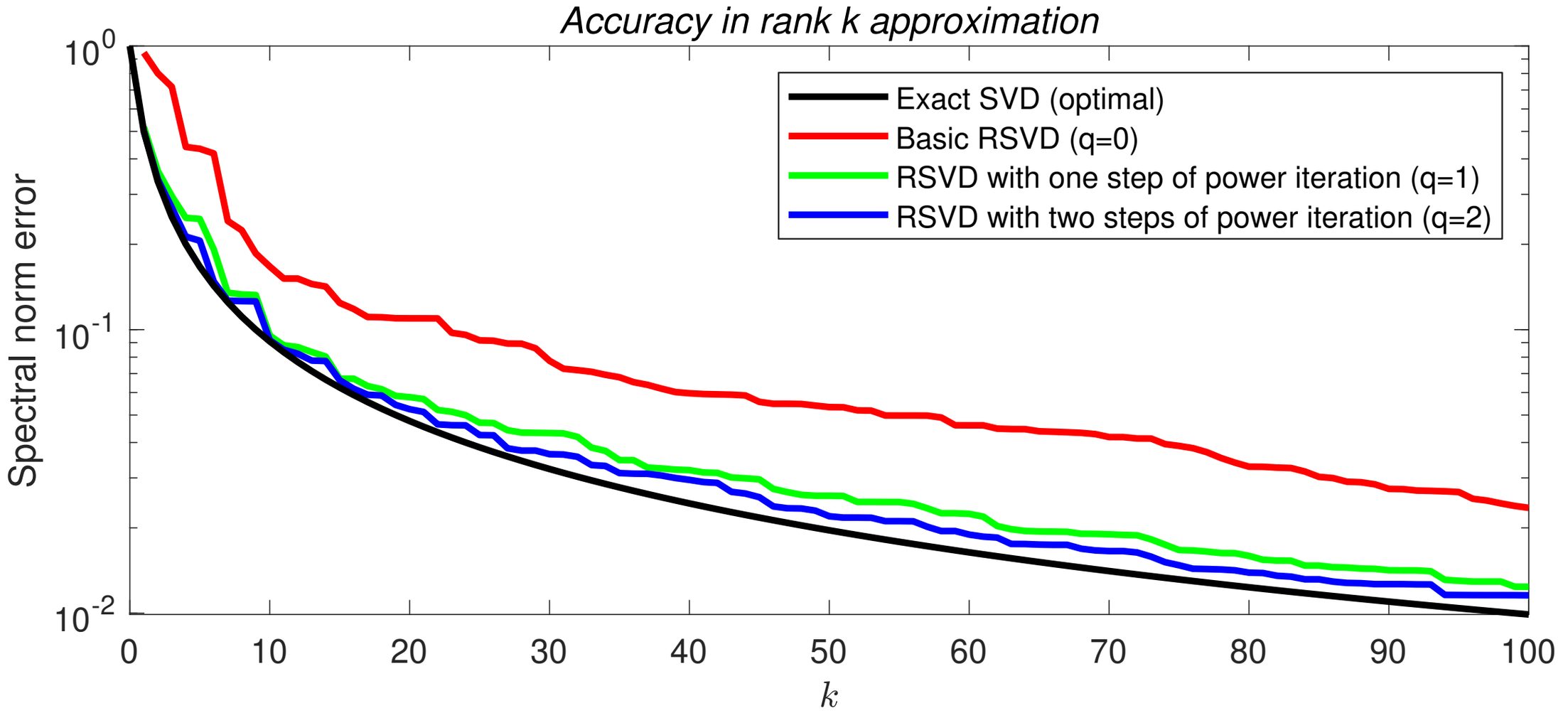
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

where $\mathbf{P}_k$ is the orthogonal projection onto the first k columns of

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

and where $\mathbf{\Omega}$ is a Gaussian random matrix. (For clarity, no oversampling is done.) The matrix $\mathbf{A}$ is an approximation to a scattering operator for a Helmholtz problem.

Randomized SVD:



The plot shows the errors from the randomized SVD. To be precise, we plot

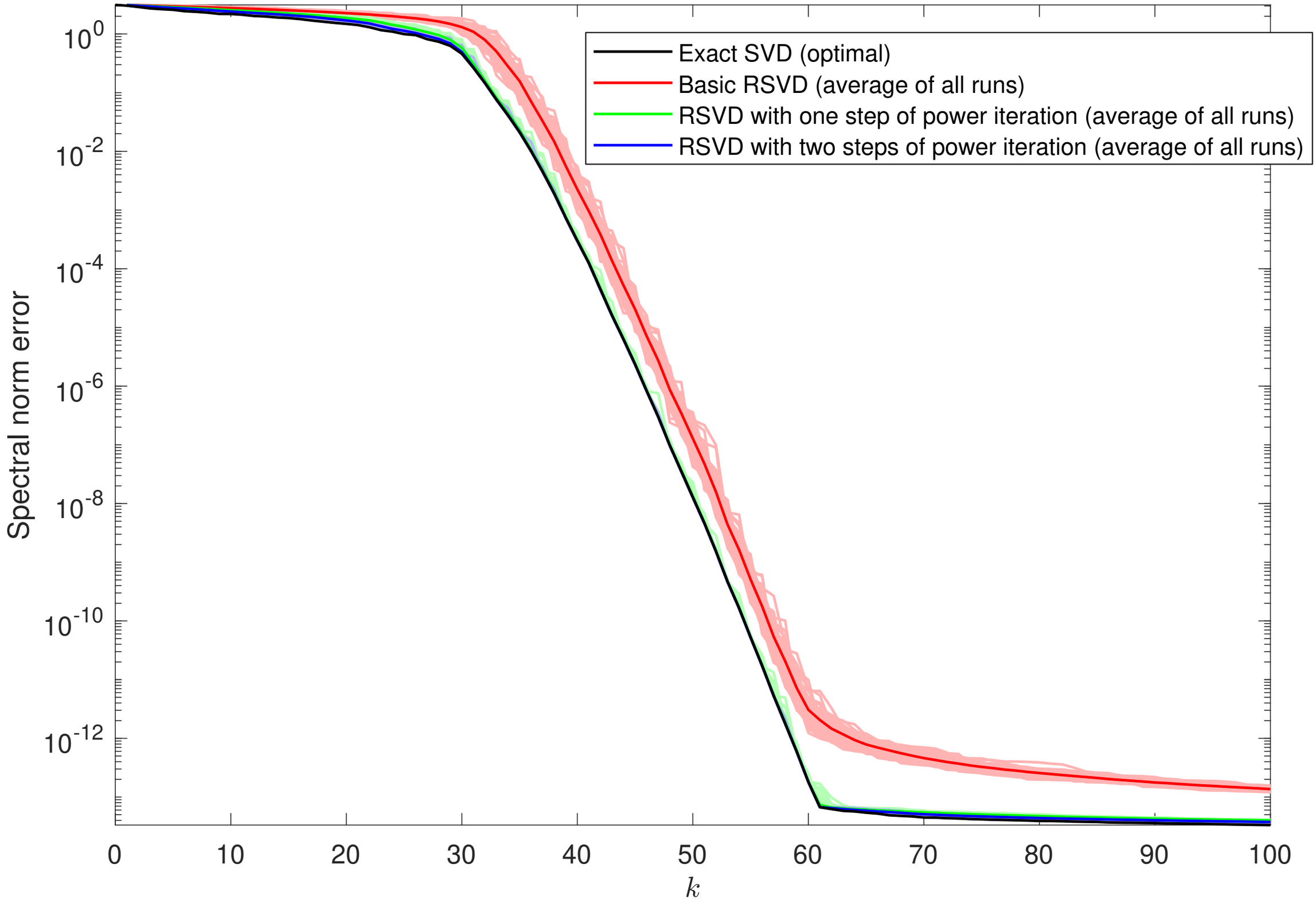
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

where $\mathbf{P}_k$ is the orthogonal projection onto the first k columns of

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

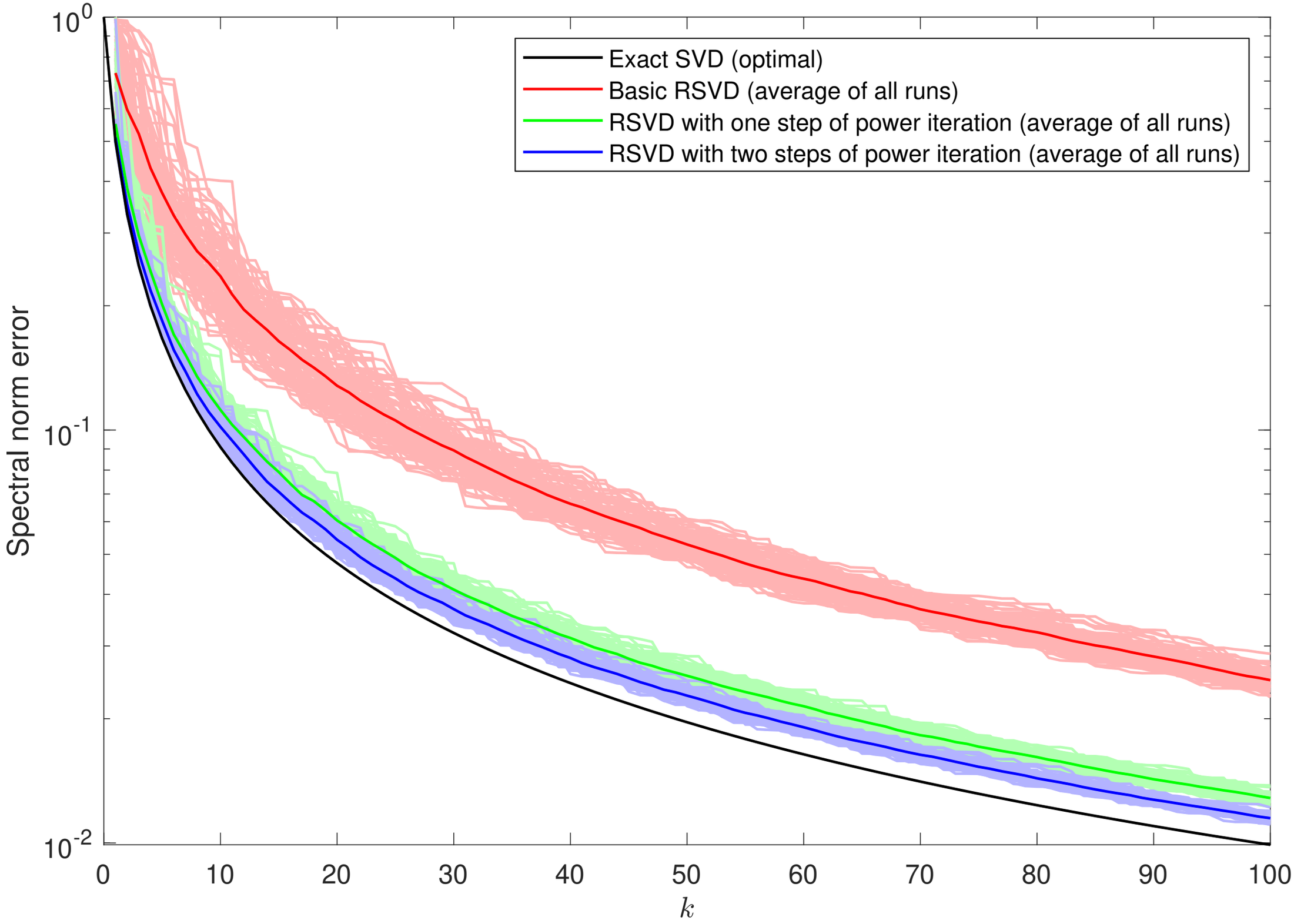
and where $\mathbf{\Omega}$ is a Gaussian random matrix. (For clarity, no oversampling is done.) The matrix $\mathbf{A}$ now has singular values that decay slowly.

Randomized SVD: The same plot as before, but now showing 100 instantiations.



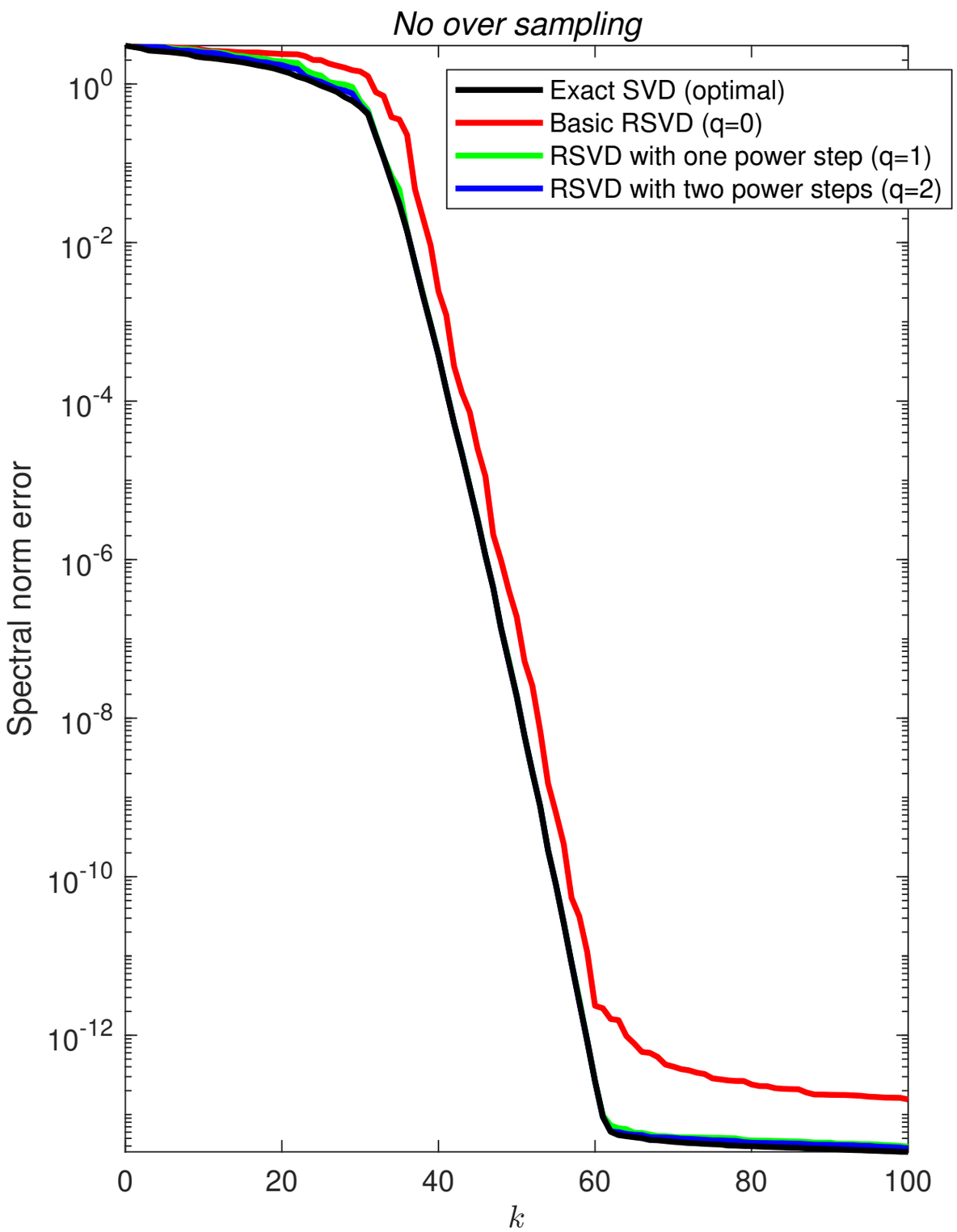
The darker lines show the mean errors across the 100 experiments.

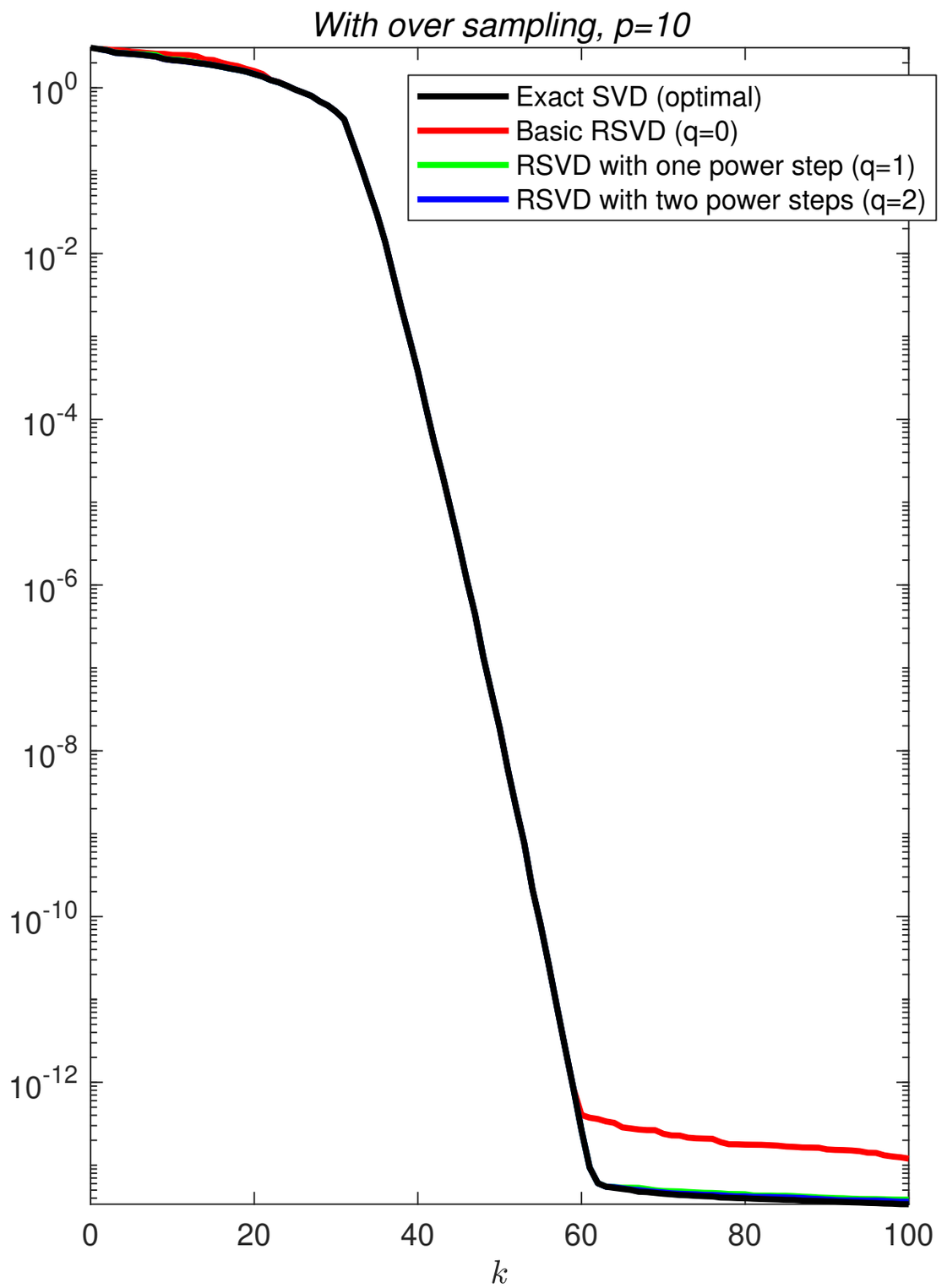
Randomized SVD: The same plot as before, but now showing 100 instantiations.



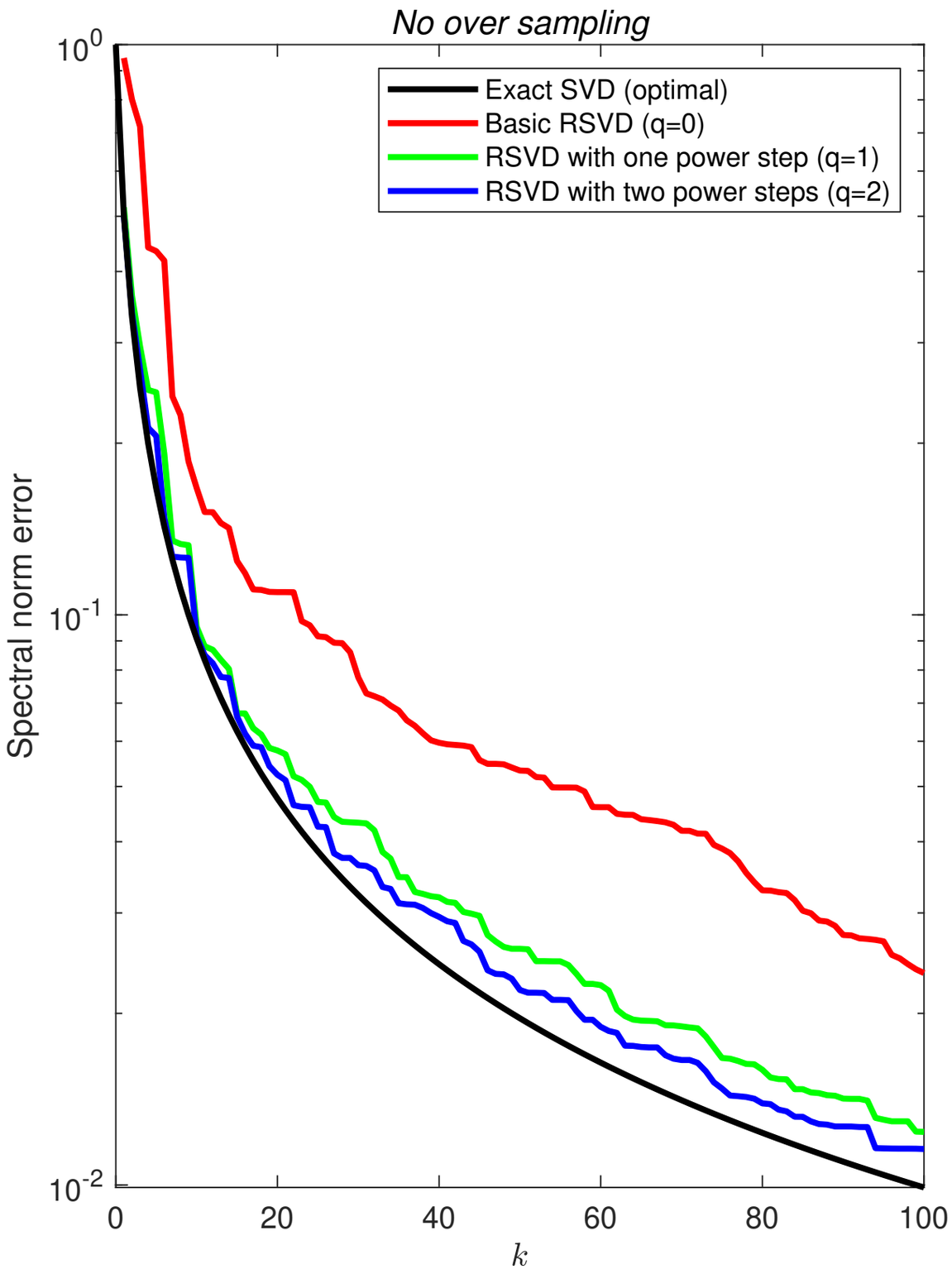
The darker lines show the mean errors across the 100 experiments.

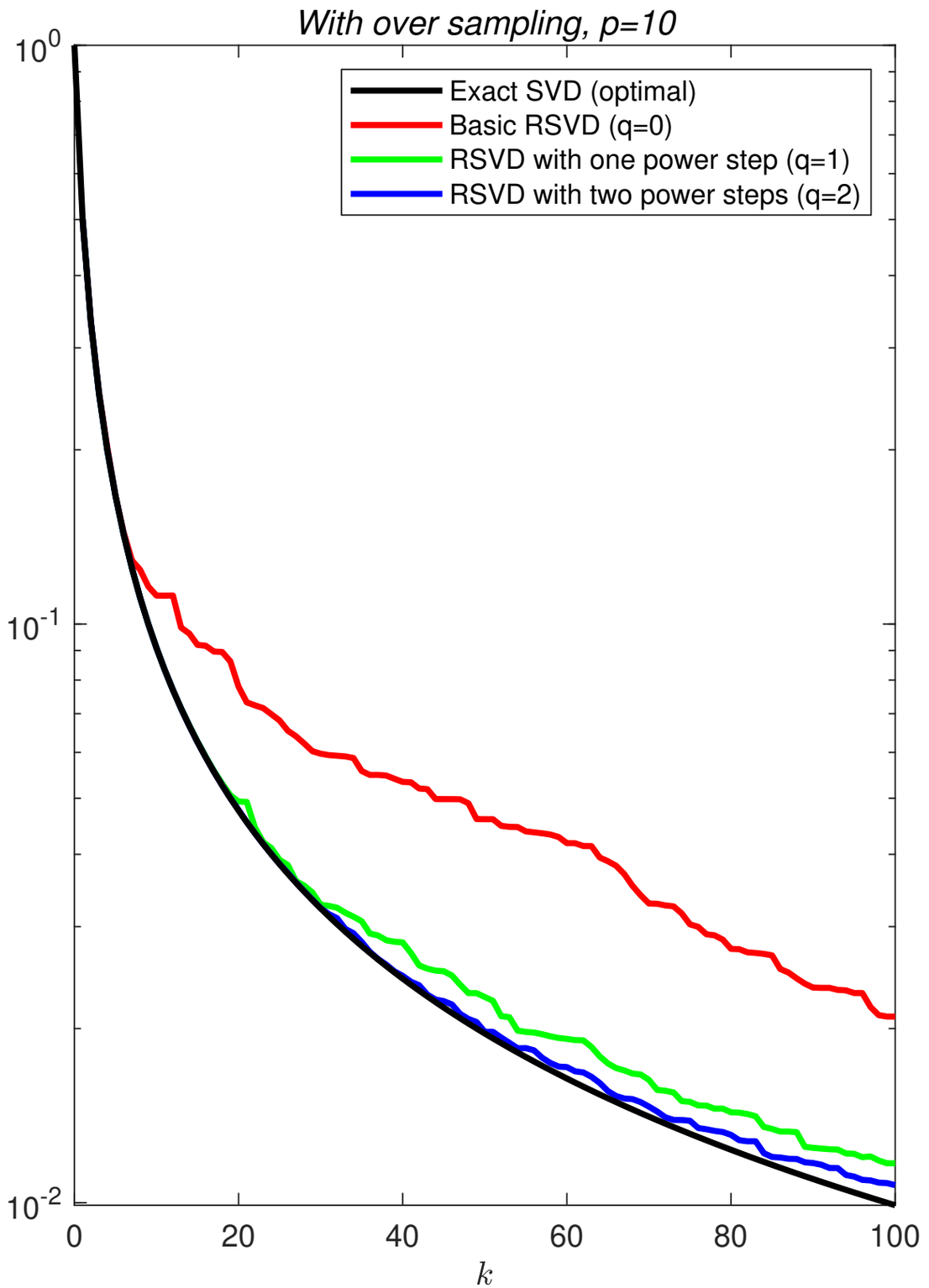
Randomized SVD: Over-sampling can further improve the optimality:





Randomized SVD: Over-sampling can further improve the optimality:





Randomized SVD:

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

Since the error in the RSVD is a random variable (it depends on the draw of $\mathbf{\Omega}$),

any theoretical analysis needs to describe the *probability distribution* of the error.

Randomized SVD:

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

Since the error in the RSVD is a random variable (it depends on the draw of $\mathbf{\Omega}$),
any theoretical analysis needs to describe the *probability distribution* of the error.

For instance, we can bound the expectation of the error:

Theorem: Let $\mathbf{A}$ be an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Let k be a target rank, and let p be an over-sampling parameter such that $p \geq 2$ and $k + p \leq \min(m, n)$. Let $\mathbf{\Omega}$ be a Gaussian random matrix of size $n \times (k+p)$ and set $\mathbf{Q} = \texttt{orth}(\mathbf{A\Omega})$. Then the average error satisfies

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

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

Randomized SVD:

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

Since the error in the RSVD is a random variable (it depends on the draw of $\boldsymbol{\Omega}$),
any theoretical analysis needs to describe the *probability distribution* of the error.

There are also bounds on the likelihood of a large deviation from the expectation.
(It turns out to decay super-exponentially fast as p increases!)

References (very incomplete!!):

- Martinsson, Rokhlin, Tygert, *Yale-CS-1361*, 2006.
- Halko, Martinsson, Tropp, *SIREV*, 2011. Survey, focus on RSVD.
- Witten/Candès, *Algorithmica*, 2015.
- Gu, *SISC*, 2015. Analysis of randomized subspace iteration.
- Musco, Musco, *NIPS*, 2015. Analysis of block Krylov methods.
- Saibaba, *SIMAX*, 2019. Accuracy of singular vectors.
- Martinsson, Tropp, *Acta Numerica*, 2020. Survey. Broader perspective.
- Nakatsukasa, *arXiv:2009.11392*, 2020. Generalized Nyström method.

Wednesday: Randomized algorithms for linear algebraic computations

1. **Randomized low rank approximation:** "Randomized singular value decomposition" or "RSVD". Relatively well established material.
2. **Variations of algorithms for low rank approximation:** Single pass and streaming algorithms. Structured random embeddings. Matrix approximation via sampling.
3. **Samples of current research directions (time permitting):** Linear solvers. Least squares problems. Block Krylov methods. Rank structured matrices.

Friday: Randomized embeddings — theory and applications

4. **Randomized embeddings:** Reducing the effective dimension of point sets. Connections to Johnson-Lindenstrauss theory. Norm estimation.
5. **Analysis of the RSVD:** Outline of probabilistic error analysis for the RSVD. The relative merits of different classes of randomized embeddings.
6. **The column/row selection problem:** Interpolatory and CUR decompositions. Pivoting in QR and LU factorizations.

Randomized SVD: A reformulation

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Randomized SVD: A reformulation

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: (Merely a restatement of what has already been described.)

Fix an over-sampling parameter p, say $p = 5$.

Draw a Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$.

Form the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.

Orthonormalize the columns of $\mathbf{Y}$ to form an ON matrix $\mathbf{Q}$.

Then $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$.

Randomized SVD: A reformulation

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: (Merely a restatement of what has already been described.)

Fix an over-sampling parameter p, say $p = 5$.

Draw a Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$.

Form the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.

Orthonormalize the columns of $\mathbf{Y}$ to form an ON matrix $\mathbf{Q}$.

Then $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$.

We seek to eliminate the intermediate matrices $\mathbf{Y}$ and $\mathbf{Q}$ from the description.

We will use the notion of a *Moore-Penrose pseudoinverse*. Brief recap:

Let $\mathbf{X}$ be an $m \times n$ matrix of rank k, with singular value decomposition

$$\begin{array}{ccccc} \mathbf{X} & = & \mathbf{U} & \mathbf{D} & \mathbf{V}^*. \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

Then the *Moore-Penrose pseudoinverse* is

$$\begin{array}{ccccc} \mathbf{X}^\dagger & = & \mathbf{V} & \mathbf{D}^{-1} & \mathbf{U}^*. \\ n \times m & & n \times k & k \times k & k \times m \end{array}$$

When $\mathbf{X}$ is invertible, $\mathbf{X}^\dagger = \mathbf{X}^{-1}$.

The property we need is: $\mathbf{XX}^\dagger$ is the orthogonal projection onto col($\mathbf{X}$).

Randomized SVD: A reformulation

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: (Merely a restatement of what has already been described.)

Fix an over-sampling parameter p, say $p = 5$.

Draw a Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$.

Form the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.

Orthonormalize the columns of $\mathbf{Y}$ to form an ON matrix $\mathbf{Q}$.

Then $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.

Using the notion of a *Moore-Penrose pseudoinverse*, we can eliminate the intermediate matrices $\mathbf{Y}$ and $\mathbf{Q}$ from the description:

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A} = \mathbf{Y}\mathbf{Y}^\dagger\mathbf{A} = (\mathbf{A\Omega})\,(\mathbf{A\Omega})^\dagger\mathbf{A}.$$

Key claim: The columns of $\mathbf{A\Omega}$ approximately span the column space of $\mathbf{A}$.

Randomized SVD: A reformulation

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: (Merely a restatement of what has already been described.)

Fix an over-sampling parameter p, say $p = 5$.

Draw a Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$.

Form the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.

Orthonormalize the columns of $\mathbf{Y}$ to form an ON matrix $\mathbf{Q}$.

Then $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$.

Using the notion of a *Moore-Penrose pseudoinverse*, we can eliminate the intermediate matrices $\mathbf{Y}$ and $\mathbf{Q}$ from the description:

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A} = \mathbf{YY}^\dagger\mathbf{A} = (\mathbf{A\Omega})\,(\mathbf{A\Omega})^\dagger\mathbf{A}.$$

Key claim: The columns of $\mathbf{A\Omega}$ approximately span the column space of $\mathbf{A}$.

Next, let us also approximate the row space: Draw a $(k+p) \times m$ Gaussian matrix $\mathbf{\Psi}$, then the rows of $\mathbf{\Psi A}$ approximately span the row space of $\mathbf{A}$. Then:

$$\mathbf{A} \approx (\mathbf{A\Omega})\,(\mathbf{A\Omega})^\dagger\mathbf{A}(\mathbf{\Psi A})^\dagger\,(\mathbf{\Psi A}).$$

Randomized SVD: A reformulation

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: (Merely a restatement of what has already been described.)

Fix an over-sampling parameter p, say $p = 5$.

Draw a Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times (k+p)}$.

Form the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.

Orthonormalize the columns of $\mathbf{Y}$ to form an ON matrix $\mathbf{Q}$.

Then $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$.

Using the notion of a *Moore-Penrose pseudoinverse*, we can eliminate the intermediate matrices $\mathbf{Y}$ and $\mathbf{Q}$ from the description:

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A} = \mathbf{YY}^\dagger\mathbf{A} = (\mathbf{A\Omega})\,(\mathbf{A\Omega})^\dagger\mathbf{A}.$$

Key claim: The columns of $\mathbf{A\Omega}$ approximately span the column space of $\mathbf{A}$.

Next, let us also approximate the row space: Draw a $(k+p) \times m$ Gaussian matrix $\mathbf{\Psi}$, then the rows of $\mathbf{\Psi A}$ approximately span the row space of $\mathbf{A}$. Then:

$$\mathbf{A} \approx (\mathbf{A\Omega})\,(\mathbf{A\Omega})^\dagger\mathbf{A}(\mathbf{\Psi A})^\dagger\,(\mathbf{\Psi A}).$$

Simplify: $\mathbf{A} \approx (\mathbf{A\Omega})\,(\mathbf{A\Omega})^\dagger\mathbf{A}(\mathbf{\Psi A})^\dagger\,(\mathbf{\Psi A}) = \cdots = (\mathbf{A\Omega})\,\big(\mathbf{\Psi A\Omega}\big)^\dagger\,(\mathbf{\Psi A}).$

Randomized SVD: Double-sided approximation ("generalized Nyström")

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m\times(k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p)\times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A\Omega}) \left(\mathbf{\Psi A\Omega}\right)^{\dagger} (\mathbf{\Psi A}) =: \mathbf{A}_{\mathrm{approx}}$.

Yuji Nakatsukasa, arXiv:2009.11392, 2020

Randomized SVD: Double-sided approximation ("generalized Nyström")

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{n \times (k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p) \times m}$.

Form approximation $\mathbf{A} \approx (\mathbf{A\Omega}) \left(\mathbf{\Psi A \Omega}\right)^{\dagger} (\mathbf{\Psi A}) =: \mathbf{A}_{\text{approx}}$.

Observation 1:

The matrix $\mathbf{A}_{\text{approx}}$ can be built *in a single pass over* $\mathbf{A}$.

In other words, we need to view each matrix entry of $\mathbf{A}$ only once.

This cannot be done using deterministic methods (as far as I know).

"Streaming" or "single-view" algorithm.

Note: *Using different over-sampling parameters for the row and column spaces is often better.*

References: Alon, Gibbons, Matias and Szegedy (2002); Woolfe, Liberty, Rokhlin, and Tygert (2008); Clarkson and Woodruff (2009); Li, Nguyen and Woodruff (2014); Boutsidis, Woodruff and Zhong (2016), Tropp, Yurtsever, Udell and Cevher (2017); Pourkamali-Anaraki and Becker (2019); Wang, Gittens and Mahoney (2019); Nakatsukasa, *arXiv:2009.11392*, 2020; Dong & Martinsson, *arXiv:2104.05877*, 2021; . . . many more . . .

Randomized SVD: Double-sided approximation ("generalized Nyström")

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m\times(k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p)\times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A\Omega}) \left(\mathbf{\Psi A\Omega}\right)^{\dagger} (\mathbf{\Psi A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A\Omega}$ and $\mathbf{\Psi A}$ requires $O(mnk)$ flops.

$O(mnk)$ matches the flop count of Gram-Schmidt, or of a Krylov method applied to a dense matrix.

Randomized SVD: Double-sided approximation ("generalized Nyström")

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m\times(k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p)\times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A\Omega})\,(\mathbf{\Psi A\Omega})^{\dagger}\,(\mathbf{\Psi A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A\Omega}$ and $\mathbf{\Psi A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that $\mathbf{A\Omega}$ *and* $\mathbf{\Psi A}$ *can be evaluated using asymptotically fewer flops!*

Randomized SVD: Double-sided approximation ("generalized Nyström")

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m\times(k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p)\times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A\Omega})\,(\mathbf{\Psi A\Omega})^{\dagger}\,(\mathbf{\Psi A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A\Omega}$ and $\mathbf{\Psi A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that $\mathbf{A\Omega}$ *and* $\mathbf{\Psi A}$ *can be evaluated using asymptotically fewer flops!*

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn\log(k))$.
- Chains of Given's rotations ("Kac's random walk"). Cost is $O(mn\log(k))$.
- "Sparse sign matrix". Place r random entries in each row of $\mathbf{\Omega}$. (Say $r \in \{2, 3, 4, 5\}$.) Cost is now $O(mn)$!

Randor

Task: F

Algoritl

Form ap

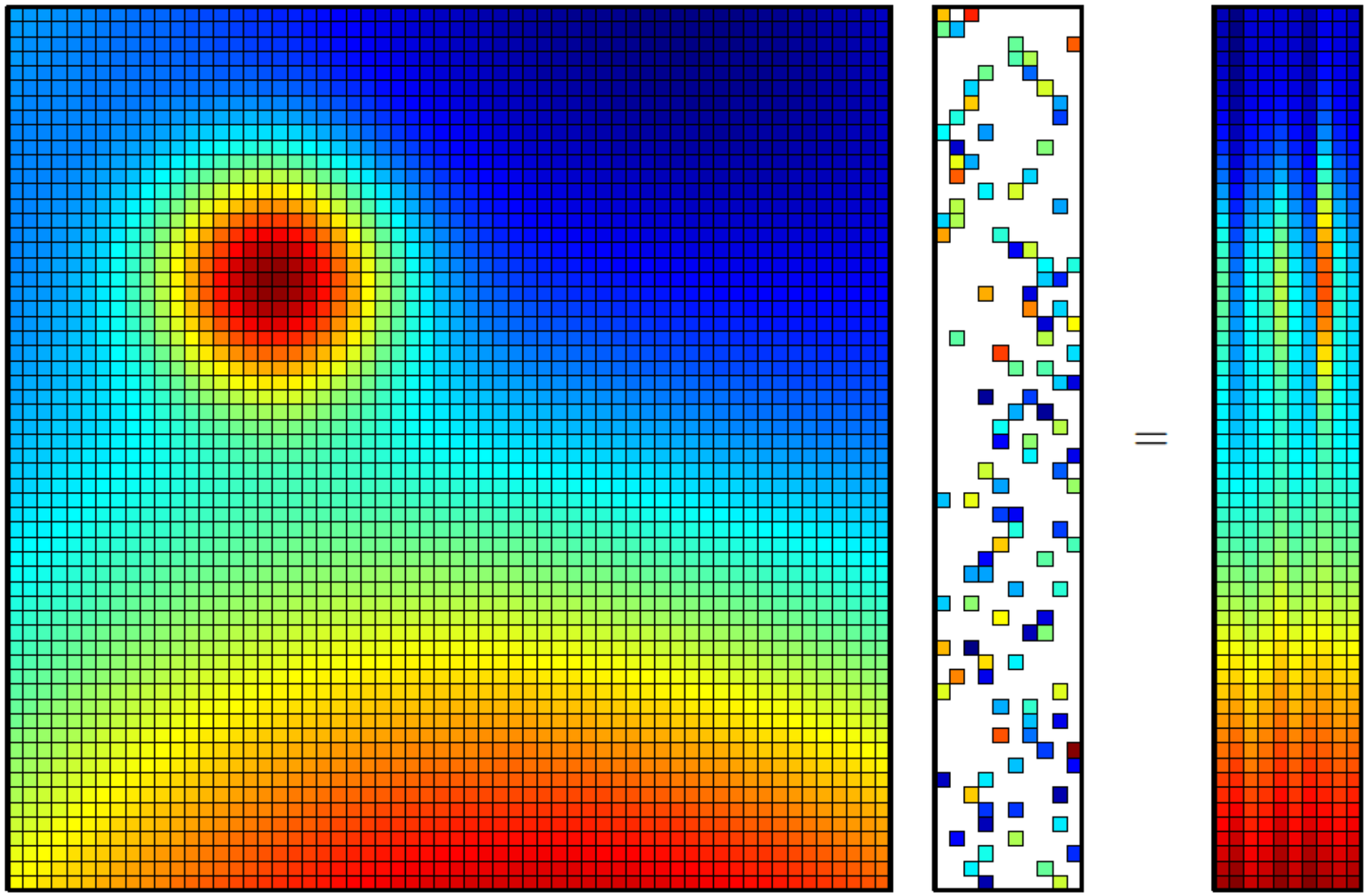
Observ

Using G

Instead

with the

- Rar
- Cha
- "Sp
  Cos



A **Ω** **AΩ**

ղ.

ps.

s **Ψ** and **Ω**

flops!

$n \log(k)$).

, 3, 4, 5}.)

The matrix **Ω** *is a sparse random matrix. Two nonzero entries are placed randomly in each row. In consequence, each column of* **A** *contributes to precisely two columns of the sample matrix* $\mathbf{Y} = \mathbf{A\Omega}$*. This structured random map has* $O(mn)$ *complexity, is easy to work with practically, and often provides good accuracy.*

Randomized SVD: Double-sided approximation ("generalized Nyström")

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Algorithm: Draw Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{m\times(k+p)}$ and $\mathbf{\Psi} \in \mathbb{R}^{(k+p)\times n}$.

Form approximation $\mathbf{A} \approx (\mathbf{A\Omega})\,(\mathbf{\Psi A\Omega})^{\dagger}\,(\mathbf{\Psi A}) =: \mathbf{A}_{\text{approx}}$.

Observation 2:

Using Gaussian random matrices, evaluating $\mathbf{A\Omega}$ and $\mathbf{\Psi A}$ requires $O(mnk)$ flops.

Instead of Gaussian random matrices, we can use *structured random matrices* $\mathbf{\Psi}$ and $\mathbf{\Omega}$ with the property that $\mathbf{A\Omega}$ *and* $\mathbf{\Psi A}$ *can be evaluated using asymptotically fewer flops!*

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn\log(k))$.
- Chains of Given's rotations ("Kac's random walk"). Cost is $O(mn\log(k))$.
- "Sparse sign matrix". Place r random entries in each row of $\mathbf{\Omega}$. (Say $r \in \{2, 3, 4, 5\}$.) Cost is now $O(mn)$!

When a structured random matrix is used, overall cost can be reduced to $O(mn + k^3)$.

Despite the pseudo-inverse, this can be done in a numerically stable way.

References: Ailon & Chazelle (2006); Liberty, Rokhlin, Tygert, and Woolfe (2006); Halko, Martinsson, Tropp (2011); Clarkson & Woodruff (2013); Meng & Mahoney (2013); Nelson & Nguyen (2013); Urano (2013); Nakatsukasa, *arXiv:2009.11392*, 2020; Dong & Martinsson, *arXiv:2104.05877*, 2021. Much subsequent work — "Fast Johnson-Lindenstrauss transform."

Low rank approximation via sampling:

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Question: Can we be even *more* aggressive? Complexity *less* than $O(mn + k^3)$?

Low rank approximation via sampling:

Task: Find a rank-k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Question: Can we be even *more* aggressive? Complexity *less* than $O(mn + k^3)$?

Sampling approach:

1. Draw vectors J and I holding k samples from the column and row indices, resp.
2. Form matrices $\mathbf{C}$ and $\mathbf{R}$ consisting of the corresponding columns and rows

$$\mathbf{C} = \mathbf{A}(:, J), \qquad \text{and} \qquad \mathbf{R} = \mathbf{A}(I, :).$$

3. Use the approximation $\mathbf{A} \approx \mathbf{CUR}$ where $\mathbf{U}$ is computed from information in $\mathbf{A}(I, J)$. ($\mathbf{U}$ should be an approximation to the optimal choice $\mathbf{U} = \mathbf{C}^\dagger \mathbf{A} \mathbf{R}^\dagger$.)

The computational profile depends crucially on the probability distribution that is used.

Uniform probabilities: Can be *very* cheap. But not reliable in the general case.

Probabilities from "leverage scores": Optimal distributions can be computed using the information in the top left and right singular vectors of $\mathbf{A}$. Quite strong theorems can be proven on the quality of this approximation. Problem: Computing the probability distribution requires computing a partial SVD. (Sometimes there are shortcuts.)

References: Drineas, Kannan & Mahoney (2006); Drineas, Mahoney & Muthukrishnan (2008); Kannan & Vempala (2017); Drineas & Mahoney (2018); Martinsson & Tropp (2020); ...

Low rank approximation via sampling: Connection to structured embeddings

Task: Find a rank k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Sampling approach (now single sided again): Draw a subset of k columns $\mathbf{C} = \mathbf{A}(:, J)$ where J is drawn at random. Consider the approximant

$$\mathbf{A}_{\text{approx}} = \mathbf{C}\mathbf{C}^{\dagger}\mathbf{A}.$$

As we have seen, this is not reliable in the general case. But it does work well for the class of matrices for which uniform sampling is optimal.

Low rank approximation via sampling: Connection to structured embeddings

Task: Find a rank k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Sampling approach (now single sided again): Draw a subset of k columns $\mathbf{C} = \mathbf{A}(:, J)$ where J is drawn at random. Consider the approximant

$$\mathbf{A}_{\text{approx}} = \mathbf{C}\mathbf{C}^{\dagger}\mathbf{A}.$$

As we have seen, this is not reliable in the general case. But it does work well for the class of matrices for which uniform sampling is optimal. *We can turn* **A** *into such a matrix!*

Low rank approximation via sampling: Connection to structured embeddings

Task: Find a rank k approximation to a given $m \times n$ matrix $\mathbf{A}$.

Sampling approach (now single sided again): Draw a subset of k columns $\mathbf{C} = \mathbf{A}(:, J)$ where J is drawn at random. Consider the approximant

$$\mathbf{A}_{\text{approx}} = \mathbf{C}\mathbf{C}^{\dagger}\mathbf{A}.$$

As we have seen, this is not reliable in the general case. But it does work well for the class of matrices for which uniform sampling is optimal. *We can turn* $\mathbf{A}$ *into such a matrix!* Let $\mathbf{U}$ be a matrix drawn from a uniform distribution on the set of $n \times n$ unitary matrices (the "Haar distribution"). Then form

$$\tilde{\mathbf{A}} = \mathbf{A}\mathbf{U}.$$

Now each column of $\tilde{\mathbf{A}}$ has exactly the same distribution! We may as well pick $J = 1 : k$, and can then pick a well behaved sample through

$$\mathbf{C} = \tilde{\mathbf{A}}(:, J) = \mathbf{A}\mathbf{U}(:, J).$$

The $n \times k$ "slice" $\mathbf{U}(:, J)$ is in a sense an optimal random embedding.

Fact: Using a Gaussian matrix is mathematically equivalent to using $\mathbf{U}(:, J)$.

Observation: A structured random embedding seeks to mimic the action of $\mathbf{U}(:, J)$.

Wednesday: Randomized algorithms for linear algebraic computations

1. **Randomized low rank approximation:** "Randomized singular value decomposition" or "RSVD". Relatively well established material.
2. **Variations of algorithms for low rank approximation:** Single pass and streaming algorithms. Structured random embeddings. Matrix approximation via sampling.
3. **Samples of current research directions (time permitting):** Linear solvers. Least squares problems. Block Krylov methods. Rank structured matrices.

Friday: Randomized embeddings — theory and applications

4. **Randomized embeddings:** Reducing the effective dimension of point sets. Connections to Johnson-Lindenstrauss theory. Norm estimation.
5. **Analysis of the RSVD:** Outline of probabilistic error analysis for the RSVD. The relative merits of different classes of randomized embeddings.
6. **The column/row selection problem:** Interpolatory and CUR decompositions. Pivoting in QR and LU factorizations.

Solving least squares problems, linear systems, . . .

- **Overdetermined least squares problems.**

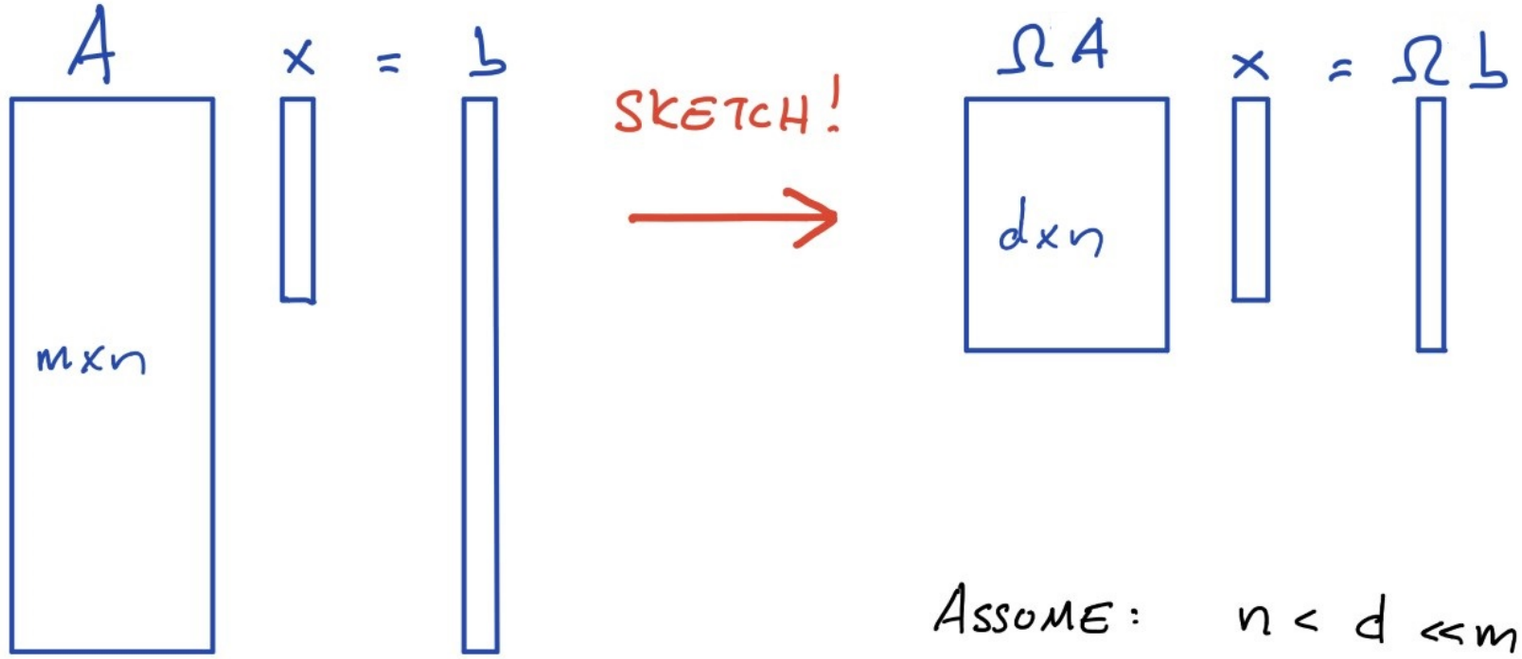
  Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$.

Solving least squares problems, linear systems, . . .

- **Overdetermined least squares problems.**

  Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$.

  Draw a random embedding $\mathbf{\Omega} \in \mathbb{R}^{d \times m}$ and construct a smaller *sketched* system.

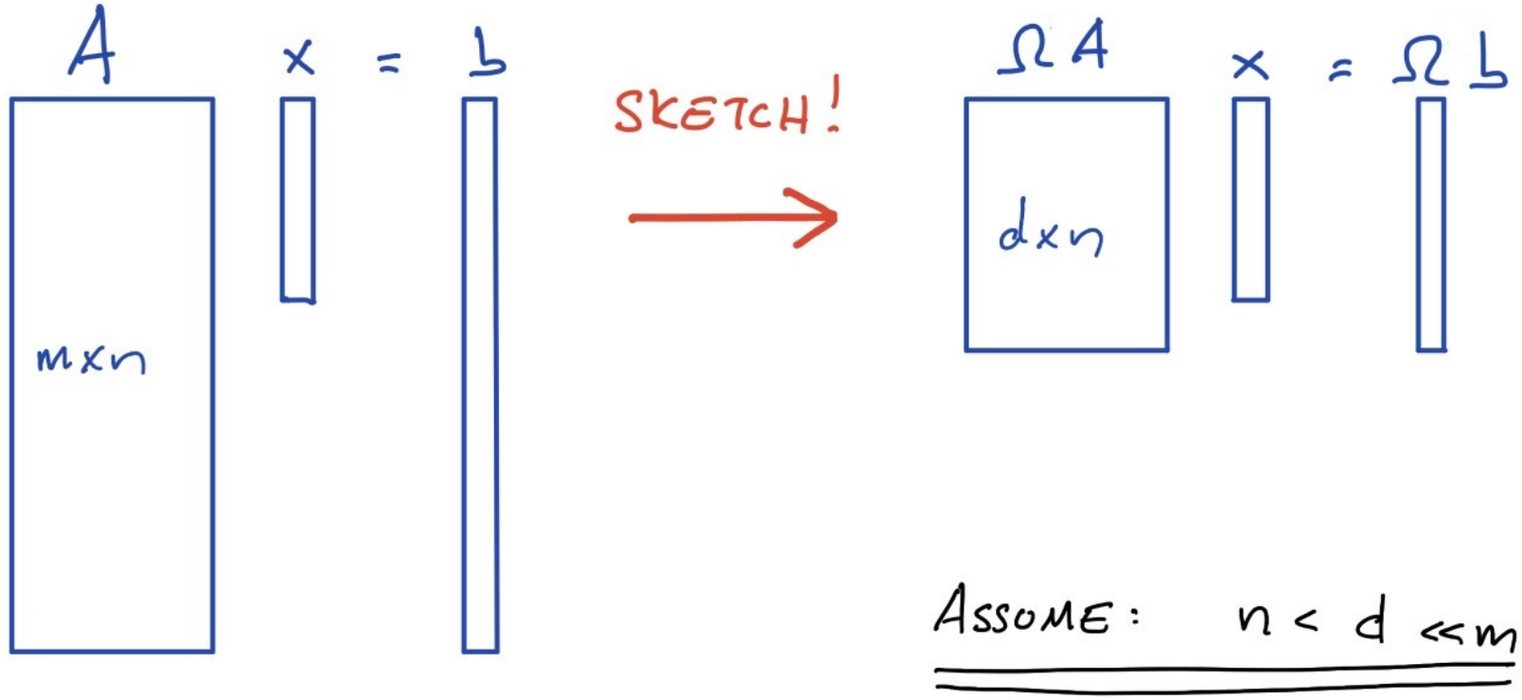


Solving least squares problems, linear systems, . . .

- **Overdetermined least squares problems.**

  Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|$.

  Draw a random embedding $\mathbf{\Omega} \in \mathbb{R}^{d \times m}$ and construct a smaller *sketched* system.



A bold approach — "sketch-to-solve":

Find the vector $\mathbf{x}$ that solves the sketched system.

A safe approach — "sketch-to-precondition":

Build a *preconditioner* $\mathbf{M} \in \mathbb{R}^{n \times n}$ by factorizing $\mathbf{\Omega A}$ so that $\mathbf{\Omega A} = \mathbf{QM}$.

Iterate on the preconditioned linear system $(\mathbf{AM}^{-1})(\mathbf{Mx}) = \mathbf{b}$.

Rokhlin/Tygert (2008), Avron/Maymounkov/Toledo (2010), many more

Solving least squares problems, linear systems, . . .

- **Overdetermined least squares problems.** *Sketch-to-precondition paradigm.*
- **Randomized Kaczmarz:** Randomization in a classical algorithm. Particularly effective when randomized embeddings are incorporated. *(Strohmer/Vershynnin 2009, Needell/Ward/Srebro 2014, Gower/Richtarik 2015, Liu/Wright 2016, . . .)*
- **Eliminating pivoting in Gaussian elimination:** D. Stott Parker showed in 1995 that you can eschew pivoting in Gaussian elimination if you first "scramble" the coefficient matrix through "pre-conditioning" via random unitary maps. Early example of fast J-L transform! Related work by Demmel/Dumitriu/Holtz (2007).
- **Graph Laplacians:** Linear systems whose coefficient matrix is a graph Laplacian can be solved using randomized methods in close to linear (in the number of edges) complexity. The idea is to compute an approximate Cholesky factorization $\mathbf{A} \approx \mathbf{C}\mathbf{C}^*$, and then use $\mathbf{C}$ as a preconditioner in conjugate gradients. Can be hybridized with ideas from nested dissection to achieve high practical speed for problems in scientific computing. *(Spielman/Teng 2004, Kyng/Sachdeva 2016, Koutis/Miller/Tolliver 2011, Livne/Brandt 2012, Spielman 2020, Chen/Liang/Biros 2020, . . .).*

Randomized methods for solving Ax = b: Graph Laplacians

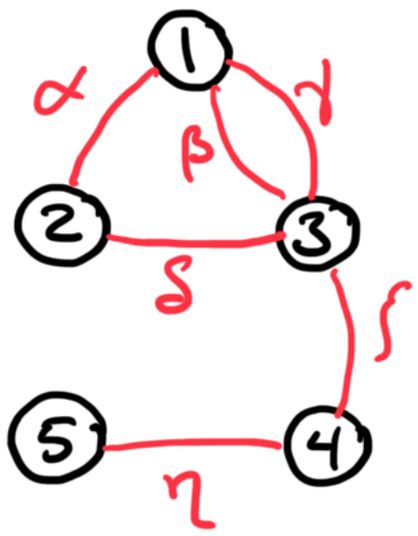
Let us consider a linear system

$$\mathbf{Ax} = \mathbf{b}$$

involving a coefficient matrix that is a *graph Laplacian* with n nodes and m edges.

- $\mathbf{A} = \mathbf{A}^* \in \mathbb{R}^{n \times n}$.
- $\mathbf{A}(i,j) \leq 0$ when $i \neq j$.
- $\mathbf{A}(i,i) = -\sum_{j \neq i} \mathbf{A}(i,j)$

We assume that the underlying graph is *connected*, in which case $\mathbf{A}$ has a 1-dimensional nullspace. We enforce that $\sum_i \mathbf{x}(i) = 0$ and $\sum_i \mathbf{b}(i) = 0$ in everything that follows.



(a) A graph with $n = 5$ vertices, and $m = 6$ edges. The conductivities of each edge is marked with a Greek letter.

$$\begin{bmatrix} \alpha+\beta+\gamma & -\alpha & -\beta-\gamma & 0 & 0 \\ -\alpha & \alpha+\delta+\zeta & -\delta & 0 & 0 \\ -\beta-\gamma & -\delta & \beta+\gamma+\delta & -\zeta & 0 \\ 0 & 0 & -\zeta & \zeta+\eta & -\eta \\ 0 & 0 & 0 & -\eta & \eta \end{bmatrix}$$

(b) The 5×5 graph Laplacian matrix associated with the graph shown in (a).

Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: Graph Laplacians

Let us consider a linear system

$$\mathbf{Ax} = \mathbf{b}$$

involving a coefficient matrix that is a *graph Laplacian* with n nodes and m edges.

Standard solution techniques:

- *Multigrid:* Works great for certain classes of matrices.

- *Cholesky:* Compute a decomposition

  $$\mathbf{A} = \mathbf{C}\mathbf{C}^*,$$

  with **C** lower triangular. Always works. Numerically stable (when pivoting is used). Can be expensive since the factor **C** typically has far more non-zero entries than **A**.

- *Incomplete Cholesky:* Compute an approximate factorisation

  $$\mathbf{A} \approx \mathbf{C}\mathbf{C}^*,$$

  where **C** is constrained to be as sparse as **A** (typically the same pattern). Then use CG to solve a system with the preconditioned coefficient matrix $\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-*}$. Can work very well, hard to analyze.

Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: Graph Laplacians

Let us consider a linear system

$$\mathbf{Ax} = \mathbf{b}$$

involving a coefficient matrix that is a *graph Laplacian* with n nodes and m edges.

Randomized solution techniques:

- *Spielman-Teng (2004):* Complexity $O(m \operatorname{poly}(\log n) \log(1/\varepsilon))$.
  Relies on graph theoretical constructs (low-stretch trees, graph sparsification, explicit expander graphs, ...). Important theoretical results.

- *Kyng-Lee-Sachdeva-Spielman (2016):* $O(m (\log n)^2)$.
  Relies on local sampling only. Much closer to a realistic algorithm.

The idea is to build an approximate sparse Cholesky factor that is accurate with high probability. For instance, the 2016 paper proposes to build factors for which

$$\frac{1}{2}\mathbf{A} \preccurlyeq \mathbf{C}\mathbf{C}^* \preccurlyeq \frac{3}{2}\mathbf{A}.$$

When this bound holds, CG converges as $O(\gamma^n)$ with $\gamma = \frac{\sqrt{3}-1}{\sqrt{3}+1} \approx 0.27$.

Sparsity is maintained by performing inexact rank-1 updates in the Cholesky procedure. As a group of edges in the graph is removed, a set of randomly drawn new edges are added, in a way that is correct *in expectation.*

Research snapshots: Approximation of kernel matrices

Consider a matrix of the form $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some kernel k and some set of points or vectors $\{\boldsymbol{x}_i\}_{i=1}^{N}$. (*Very* loose definition ...)

Matrices of this type arise frequently in both data analysis and in scientific computing.

It is generally speaking impossible to form all entries of $\mathbf{A}$ explicitly. Sampling methods become essential.

Research snapshots: Approximation of kernel matrices

Consider a matrix of the form $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some kernel k and some set of points or vectors $\{\boldsymbol{x}_i\}_{i=1}^N$. (*Very* loose definition ...)

Matrices of this type arise frequently in both data analysis and in scientific computing.

It is generally speaking impossible to form all entries of $\mathbf{A}$ explicitly. Sampling methods become essential.

Option 1: Approximate A as a matrix of global low rank

Typically leads to low accuracy, but can be "good enough" for pre-conditioning, for capturing essential features in learning problems, etc.

The types of random embeddings we have discussed in this talk that intermix all matrix elements are rarely applicable. Instead, sampling is necessary.

Example — Kernel Ridge Regression: Need to solve a linear system $\big(\mathbf{A} + \mu\mathbf{I}\big)\mathbf{x} = \mathbf{b}$ where $\mathbf{A}$ is a kernel matrix. Techniques based on Nyström approximation combined with randomized sampling have proven highly effective. *(Alaoui/Mahoney 2015, Avron/Clarkson/Woodruff 2017, Musco/Musco 2017, Rudi/Calandriello/Carratino/Rosasco 2018, ...)*

Research snapshots: Approximation of kernel matrices

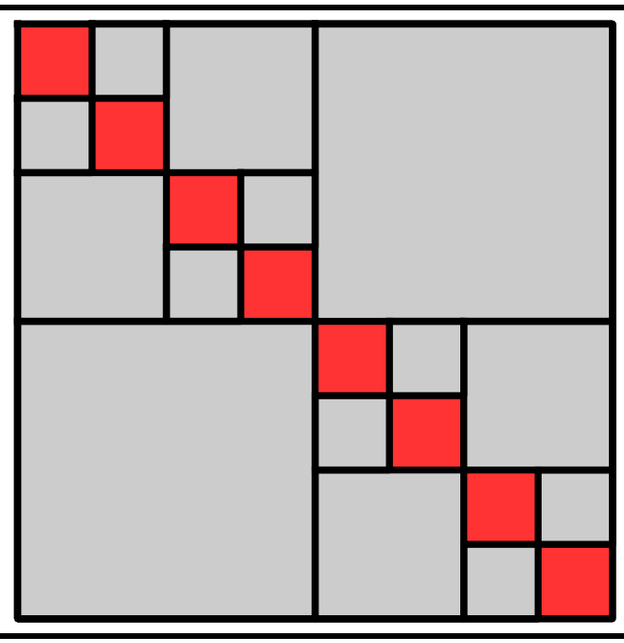
Consider a matrix of the form $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some kernel k and some set of points or vectors $\{\boldsymbol{x}_i\}_{i=1}^N$. (*Very* loose definition ...)

Matrices of this type arise frequently in both data analysis and in scientific computing.

It is generally speaking impossible to form all entries of $\mathbf{A}$ explicitly. Sampling methods become essential.

Option 2: Tessellate A into blocks that each have low rank — "$O(n)$ data"

A representative tessellation of a rank-structured matrix. Each off-diagonal block (gray) has low numerical rank. The diagonal blocks (red) are full rank, but are small in size. Matrices of this type allow efficient matrix-vector multiplication, matrix inversion, etc.



Keywords: $\mathcal{H}$-matrices, Hierarchically Semi-Separable matrices, Hierarchically Block Separable matrices, Hierarchically off-diagonal low rank matrices, Fast Multipole Method, Barnes-Hut, ...

Research snapshots: Approximation of kernel matrices

Consider a matrix of the form $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some kernel k and some set of points or vectors $\{\boldsymbol{x}_i\}_{i=1}^N$. (*Very* loose definition . . .)

Matrices of this type arise frequently in both data analysis and in scientific computing.

It is generally speaking impossible to form all entries of $\mathbf{A}$ explicitly. Sampling methods become essential.

Option 2: Tessellate A into blocks that each have low rank — "$O(n)$ data"

Randomized sampling strategies can be used to build a data sparse representation.

In scientific computing, we sometimes have technique for evaluating *global* matrix-vector products. In such cases, randomized embedding techniques do apply, and can lead to high accuracy approximations to the matrix.

(Martinsson 2011, March/Xiao/Biros 2015, Ambikasaran/Foreman-Mackey/Greengard/Hogg/O'Neil 2015, Ghysels/Li/Gorman/Rouet 2016, Yu/Levitt/Reiz/Biros 2017, Rebrova/Chávez/Liu/Ghysels/Li 2018, Geoga/Anitescu/Stein 2019, . . .)

Research snapshots: Randomized Krylov methods

Given an $n \times n$ matrix $\mathbf{A}$ (say symmetric), how build a subspace that captures its range?

Option 1: Classical Krylov method

Start with one random vector ω, and use $V = \text{span}\{\omega, \mathbf{A}\omega, \mathbf{A}^2\omega, \ldots, \mathbf{A}^{k-1}\omega\}$.

Option 2: Basic RSVD

Start with k random vectors $\{\omega_j\}_{j=1}^{k}$, and use $V = \text{span}\{\mathbf{A}\omega_1, \mathbf{A}\omega_2, \ldots, \mathbf{A}\omega_k\}$.

Research snapshots: Randomized Krylov methods

Given an $n \times n$ matrix $\mathbf{A}$ (say symmetric), how build a subspace that captures its range?

Option 1: Classical Krylov method

Start with one random vector ω, and use $V = \text{span}\{\omega, \mathbf{A}\omega, \mathbf{A}^2\omega, \ldots, \mathbf{A}^{k-1}\omega\}$.

Option 2: Basic RSVD

Start with k random vectors $\{\omega_j\}_{j=1}^k$, and use $V = \text{span}\{\mathbf{A}\omega_1, \mathbf{A}\omega_2, \ldots, \mathbf{A}\omega_k\}$.

Intermediate options: In between is a rich design space. We discussed "powering" in the context of the RSVD. You can also consider variations of block Krylov methods, where we start with a tall thin random matrix $\mathbf{\Omega}$, and then use

$$V = \text{span}\{\mathbf{A\Omega}, \mathbf{A}^2\mathbf{\Omega}, \ldots, \mathbf{A}^q\mathbf{\Omega}\}.$$

How choose parameters to optimize storage vs. flops vs. matrix accesses? What errors would you expect? How avoid numerical instability? Etc.

Musco & Musco; Tropp; Wang, Zhang, Zhang; Yuan, Gu, Li; Drineas, Ipsen, Kontopoulou, Magdon-Ismail; …

Wednesday: Randomized algorithms for linear algebraic computations

1. **Randomized low rank approximation:** "Randomized singular value decomposition" or "RSVD". Relatively well established material.
2. **Variations of algorithms for low rank approximation:** Single pass and streaming algorithms. Structured random embeddings. Matrix approximation via sampling.
3. **Samples of current research directions (time permitting):** Linear solvers. Least squares problems. Block Krylov methods. Rank structured matrices.

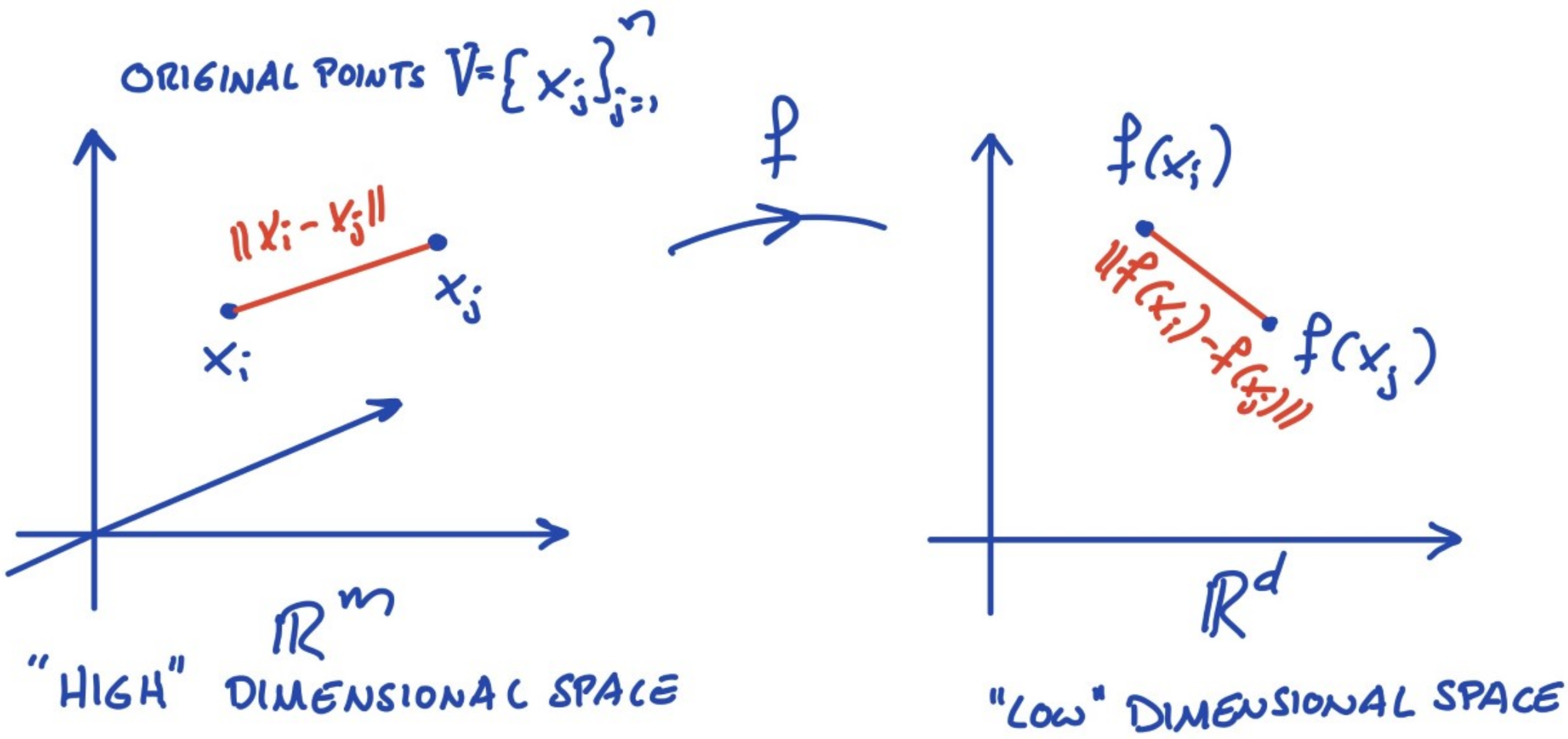
Friday: Randomized embeddings — theory and applications

4. **Randomized embeddings:** Reducing the effective dimension of point sets. Connections to Johnson-Lindenstrauss theory. Norm estimation.
5. **Analysis of the RSVD:** Outline of probabilistic error analysis for the RSVD. The relative merits of different classes of randomized embeddings.
6. **The column/row selection problem:** Interpolatory and CUR decompositions. Pivoting in QR and LU factorizations.

Randomized embeddings: What are they?

Randomized embeddings: What are they?

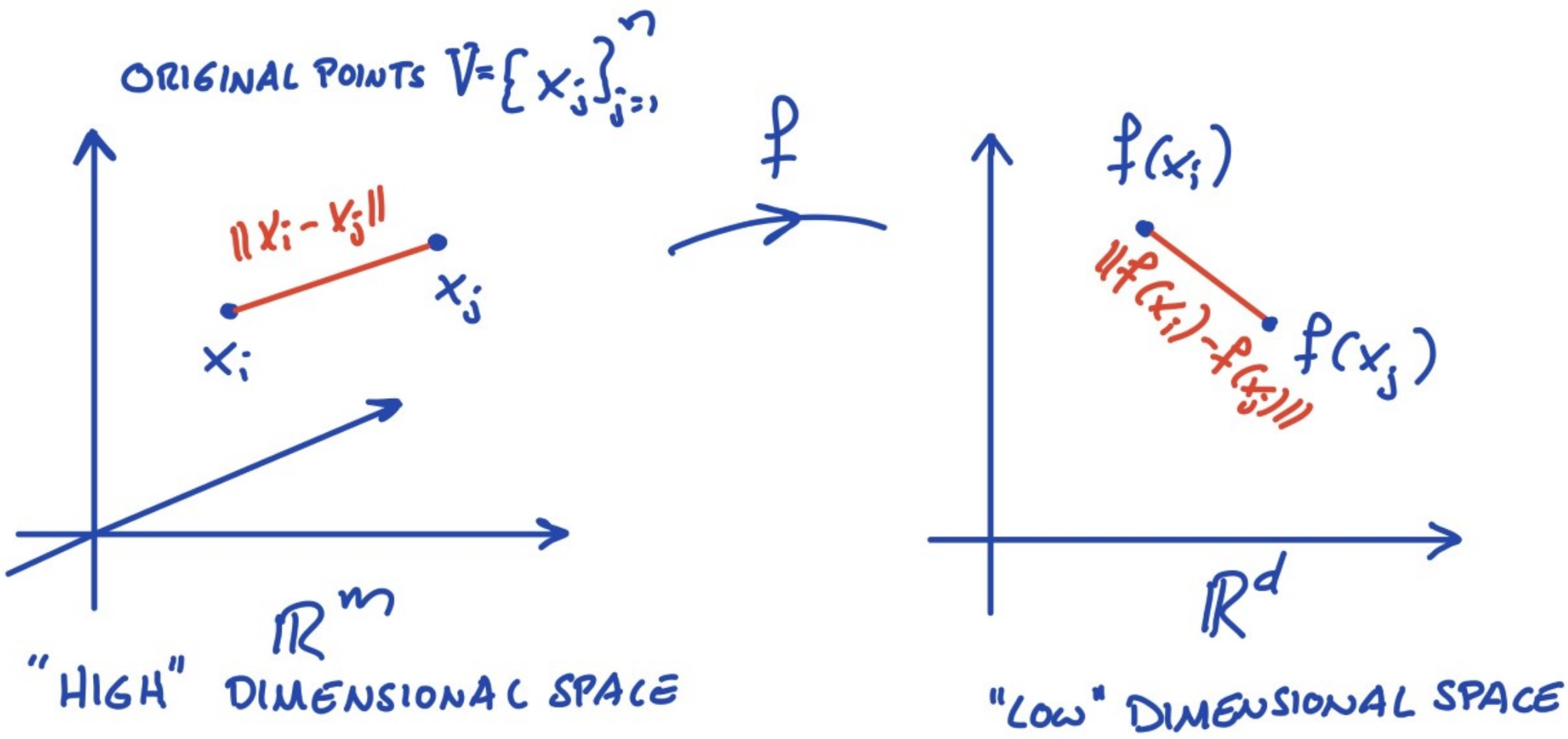
Let $V = \{\mathbf{x}_j\}_{j=1}^n$ be a set of points in $\mathbb{R}^m$, and let $f : V \to \mathbb{R}^d$ be a map, where $d < m$.
Think of m as a "large" dimension, and d as a "small" dimension.



We say that f is an *embedding* if: $\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\| \approx \|\mathbf{x}_i - \mathbf{x}_j\|, \qquad \forall\, i,j \in \{1,2,\ldots,n\}.$

Randomized embeddings: What are they?

Let $V = \{\mathbf{x}_j\}_{j=1}^n$ be a set of points in $\mathbb{R}^m$, and let $f : V \to \mathbb{R}^d$ be a map, where $d < m$.
Think of m as a "large" dimension, and d as a "small" dimension.



We say that f is an *embedding* if: $\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\| \approx \|\mathbf{x}_i - \mathbf{x}_j\|, \qquad \forall\, i,j \in \{1,2,\ldots,n\}.$

Lemma [Johnson-Lindenstrauss]: For $d \sim \log(n)$, there exists a well-behaved embedding f that "approximately" preserves distances.

Randomized embeddings: Applications

Consider a case where we are given a set $\{\mathbf{a}^{(j)}\}_{j=1}^{n}$ of points in $\mathbb{R}^m$, where m is very large. Now suppose that we need to solve tasks such as:

- Suppose the points almost live on a linear subspace of (small) dimension k. Find a basis for the "best" subspace. (Principal component analysis.)
- Given k, find the subset of k vectors with maximal spanning volume.
- Suppose the points almost live on a low-dimensional nonlinear manifold. Find a parameterization of the manifold.
- Given k, find for each vector $\mathbf{a}^{(j)}$ its k closest neighbors.
- Partition the points into clusters.

(Note: Some problems have well-defined solutions; some do not. The first can be solved with algorithms with moderate complexity; some are combinatorially hard.)

If we can embed the given set of points in a lower dimensional space, while approximately preserving distances, then a variety of algorithms for solving these problems become available.

Randomized embeddings: Gaussian embeddings

We will next describe how random matrices can be used to build embeddings.

Recall: We say that an $m \times n$ matrix $\mathbf{G}$ is a "Gaussian matrix" if each entry is drawn independently from a standard normal distribution. A "Gaussian vector" is defined analogously.

Warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Randomized embeddings: Gaussian embeddings

We will next describe how random matrices can be used to build embeddings.

Recall: We say that an $m \times n$ matrix $\mathbf{G}$ is a "Gaussian matrix" if each entry is drawn independently from a standard normal distribution. A "Gaussian vector" is defined analogously.

Warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Note: This problem is very simple to solve directly!

The purpose of the randomized method we will discuss is to introduce concepts.

Randomized embeddings: Gaussian embeddings

We will next describe how random matrices can be used to build embeddings.

Recall: We say that an $m \times n$ matrix $\mathbf{G}$ is a "Gaussian matrix" if each entry is drawn independently from a standard normal distribution. A "Gaussian vector" is defined analogously.

Warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Algorithm:

1. Draw a Gaussian vector $\mathbf{g} \in \mathbb{R}^n$.
2. Set $y = \mathbf{g} \cdot \mathbf{x}$ and use the estimate $\|\mathbf{x}\|^2 \approx y^2$.

Claim: The expectation of y^2 equals $\|\mathbf{x}\|^2$. (I.e. y^2 is an "unbiased estimate" for $\|\mathbf{x}\|^2$.)

Randomized embeddings: Gaussian embeddings

We will next describe how random matrices can be used to build embeddings.

Recall: We say that an $m \times n$ matrix $\mathbf{G}$ is a "Gaussian matrix" if each entry is drawn independently from a standard normal distribution. A "Gaussian vector" is defined analogously.

Warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Algorithm:

1. Draw a Gaussian vector $\mathbf{g} \in \mathbb{R}^n$.
2. Set $y = \mathbf{g} \cdot \mathbf{x}$ and use the estimate $\|\mathbf{x}\|^2 \approx y^2$.

Claim: The expectation of y^2 equals $\|\mathbf{x}\|^2$. (I.e. y^2 is an "unbiased estimate" for $\|\mathbf{x}\|^2$.)

$$\mathbb{E}[y^2] = \mathbb{E}[\Big(\sum_{j=1}^{n} g_j x_j\Big)^2] = \mathbb{E}[\sum_{i,j=1}^{n} g_i g_j x_i x_j] = \sum_{i,j=1}^{n} \mathbb{E}[g_i g_j]\, x_i x_j = \sum_{i,j=1}^{n} \delta_{i,j}\, x_i x_j = \sum_{j=1}^{n} x_j^2.$$

Randomized embeddings: Gaussian embeddings

We will next describe how random matrices can be used to build embeddings.

Recall: We say that an $m \times n$ matrix $\mathbf{G}$ is a "Gaussian matrix" if each entry is drawn independently from a standard normal distribution. A "Gaussian vector" is defined analogously.

Warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Algorithm:

1. Draw a Gaussian vector $\mathbf{g} \in \mathbb{R}^n$.
2. Set $y = \mathbf{g} \cdot \mathbf{x}$ and use the estimate $\|\mathbf{x}\|^2 \approx y^2$.

Claim: The expectation of y^2 equals $\|\mathbf{x}\|^2$. (I.e. y^2 is an "unbiased estimate" for $\|\mathbf{x}\|^2$.)

$$\mathbb{E}[y^2] = \mathbb{E}[\Big(\sum_{j=1}^{n} g_j x_j\Big)^2] = \mathbb{E}[\sum_{i,j=1}^{n} g_i g_j x_i x_j] = \sum_{i,j=1}^{n} \mathbb{E}[g_i g_j]\, x_i x_j = \sum_{i,j=1}^{n} \delta_{i,j}\, x_i x_j = \sum_{j=1}^{n} x_j^2.$$

Problem: y^2 is not a practically useful estimate for $\|\mathbf{x}\|^2$, since the variance is large.

$$\begin{array}{ccccc} (\mathbf{g} \cdot \mathbf{x})^2 & = & (\cos\theta)^2 & \times & \|\mathbf{g}\|^2 \times \|\mathbf{x}\|^2 \\ & & \sim \frac{1}{n} & & \sim n \end{array}$$

Randomized embeddings: Gaussian embeddings

We will next describe how random matrices can be used to build embeddings.

Recall: We say that an $m \times n$ matrix $\mathbf{G}$ is a "Gaussian matrix" if each entry is drawn independently from a standard normal distribution. A "Gaussian vector" is defined analogously.

Warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Algorithm:

1. Draw a Gaussian vector $\mathbf{g} \in \mathbb{R}^n$.
2. Set $y = \mathbf{g} \cdot \mathbf{x}$ and use the estimate $\|\mathbf{x}\|^2 \approx y^2$.

Claim: The expectation of y^2 equals $\|\mathbf{x}\|^2$. (I.e. y^2 is an "unbiased estimate" for $\|\mathbf{x}\|^2$.)

$$\mathbb{E}[y^2] = \mathbb{E}[\Big(\sum_{j=1}^{n} g_j x_j\Big)^2] = \mathbb{E}[\sum_{i,j=1}^{n} g_i g_j x_i x_j] = \sum_{i,j=1}^{n} \mathbb{E}[g_i g_j]\, x_i x_j = \sum_{i,j=1}^{n} \delta_{i,j}\, x_i x_j = \sum_{j=1}^{n} x_j^2.$$

Problem: y^2 is not a practically useful estimate for $\|\mathbf{x}\|^2$, since the variance is large. Any one experiment is likely to give a substantial error.

Question: How can we improve the quality of the estimate?

Randomized embeddings: Gaussian embeddings

Recall warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Improved algorithm:

1. Pick a positive integer d. (As d grows, cost grows, and accuracy improves.)
2. Draw a $d \times n$ Gaussian matrix $\mathbf{G}$.
3. Set $\mathbf{y} = \frac{1}{\sqrt{d}}\mathbf{G}\mathbf{x}$ and use $\|\mathbf{x}\|^2 \approx \|\mathbf{y}\|^2$.

Claim: The random variable $\|\mathbf{y}\|^2$ is an "unbiased" estimate for $\|\mathbf{x}\|^2$:

Proof: An elementary computation shows that

$$\|\mathbf{y}\|^2 = \sum_{j=1}^{d} y_j^2 = \frac{1}{d}\sum_{j=1}^{d} (\mathbf{G}(j,:)\mathbf{x})^2.$$

Now observe that $\mathbf{G}(j,:)$ is a Gaussian vector, so by the proof on the previous slide, the random variable $(\mathbf{G}(j,:)\mathbf{x})^2$ has expectation $\|\mathbf{x}\|^2$.

Since $\|\mathbf{y}\|^2$ is the average of d random variables, each with expectation $\|\mathbf{x}\|^2$, its expectation is also $\|\mathbf{x}\|^2$.

Randomized embeddings: Gaussian embeddings

Recall warm up problem: Given $\mathbf{x} \in \mathbb{R}^n$, estimate its ℓ^2 norm.

Improved algorithm:

1. Pick a positive integer d. (As d grows, cost grows, and accuracy improves.)
2. Draw a $d \times n$ Gaussian matrix $\mathbf{G}$.
3. Set $\mathbf{y} = \frac{1}{\sqrt{d}}\mathbf{G}\mathbf{x}$ and use $\|\mathbf{x}\|^2 \approx \|\mathbf{y}\|^2$.

Claim: The random variable $\|\mathbf{y}\|^2$ is an "unbiased" estimate for $\|\mathbf{x}\|^2$:

Proof: An elementary computation shows that

$$\|\mathbf{y}\|^2 = \sum_{j=1}^{d} y_j^2 = \frac{1}{d}\sum_{j=1}^{d} (\mathbf{G}(j,:)\mathbf{x})^2.$$

Now observe that $\mathbf{G}(j,:)$ is a Gaussian vector, so by the proof on the previous slide, the random variable $(\mathbf{G}(j,:)\mathbf{x})^2$ has expectation $\|\mathbf{x}\|^2$.

Since $\|\mathbf{y}\|^2$ is the average of d random variables, each with expectation $\|\mathbf{x}\|^2$, its expectation is also $\|\mathbf{x}\|^2$.

Important: *The variance of $\|\mathbf{y}\|^2$ goes to zero as d grows.*

Randomized embeddings: Gaussian embeddings

We are now ready to return to our *real* question of how to embed a set $V = \{\mathbf{x}_j\}_{j=1}^n$ into a lower dimensional space.

For a positive integer d, draw a $d \times n$ Gaussian matrix $\mathbf{G}$ and set $f(\mathbf{x}) = \frac{1}{\sqrt{d}}\mathbf{G}\mathbf{x}$.

We know that $\mathbb{E}\left[\|f(\mathbf{x}) - f(\mathbf{y})\|^2\right] = \|\mathbf{x} - \mathbf{y}\|^2$, and that as d grows, the probability distribution will concentrate around the expectation.

Claim: Given an $\varepsilon > 0$, pick a positive integer

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

Then with positive probability, we have

$$(1-\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|^2 \leq (1+\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\|^2, \qquad \forall\, i,j \in \{1,2,\ldots,n\}. \tag{2}$$

Sketch of proof:

(1) Establish that $\frac{d}{\|\mathbf{x}\|^2}\|\mathbf{G}\mathbf{x}\|^2$ has a χ^2 distribution of degree d.

(2) Use known properties of the χ^2 distribution.

(3) Apply a simple union bound.

Randomized embeddings: Gaussian embeddings

To summarize, we have outlined a proof for:

Lemma [Johnson-Lindenstrauss]: *Let ε be a real number such that $\varepsilon \in (0, 1)$, let n be a positive integer, and let d be an integer such that*

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

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

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

Randomized embeddings: Gaussian embeddings

To summarize, we have outlined a proof for:

Lemma [Johnson-Lindenstrauss]: *Let ε be a real number such that $\varepsilon \in (0, 1)$, let n be a positive integer, and let d be an integer such that*

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

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

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

Practical problem: You have two bad choices:

(1) Pick a small ε; then you get small distortions, but a huge d since $d \sim \frac{8}{\varepsilon^2} \log(n)$.

(2) Pick ε that is not close to 0; then distortions are large.

The behavior is analogous to classical Monte Carlo methods.

Matrix approximation by sampling

Suppose that $\mathbf{A} = \sum_{t=1}^{T} \mathbf{A}_t$ where each $\mathbf{A}_t$ is “simple” in some sense.

Matrix approximation by sampling

Suppose that $\mathbf{A} = \sum_{t=1}^{T} \mathbf{A}_t$ where each $\mathbf{A}_t$ is "simple" in some sense.

Example: Sparse matrix written as a sum over its nonzero entries

$$\underbrace{\begin{bmatrix} 5 & -2 & 0 \\ 0 & 0 & -3 \\ 1 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}} = \underbrace{\begin{bmatrix} 5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_1} + \underbrace{\begin{bmatrix} 0 & -2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_2} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_3} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_4}$$

Example: Each $\mathbf{A}_i$ could be a column of the matrix

$$\underbrace{\begin{bmatrix} 5 & -2 & 7 \\ 1 & 3 & -3 \\ 1 & -1 & 1 \end{bmatrix}}_{=\mathbf{A}} = \underbrace{\begin{bmatrix} 5 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{=\mathbf{A}_1} + \underbrace{\begin{bmatrix} 0 & -2 & 0 \\ 0 & 3 & 0 \\ 0 & -1 & 0 \end{bmatrix}}_{=\mathbf{A}_2} + \underbrace{\begin{bmatrix} 0 & 0 & 7 \\ 0 & 0 & -3 \\ 0 & 0 & 1 \end{bmatrix}}_{=\mathbf{A}_3}.$$

Example: Matrix-matrix multiplication broken up as a sum of rank-1 matrices:

$$\mathbf{A} = \mathbf{BC} = \sum_{t} \mathbf{B}(:,t)\mathbf{C}(t,:).$$

Matrix approximation by sampling

Suppose that $\mathbf{A} = \sum_{t=1}^{T} \mathbf{A}_t$ where each $\mathbf{A}_t$ is "simple" in some sense.

Let $\{p_t\}_{t=1}^{T}$ be a probability distribution on the index vector $\{1, 2, \ldots, T\}$.
Draw an index $t \in \{1, 2, \ldots, T\}$ according to the probability distribution given, and set

$$\mathbf{X} = \frac{1}{p_t}\mathbf{A}_t.$$

Then from the definition of the expectation, we have

$$\mathbb{E}[\mathbf{X}] = \sum_{t=1}^{T} p_t \times \frac{1}{p_t}\mathbf{A}_t = \sum_{t=1}^{T} \mathbf{A}_t = \mathbf{A},$$

so $\mathbf{X}$ is an unbiased estimate of $\mathbf{A}$.

Clearly, a single draw is not a good approximation — unrepresentative, *large variance*.

Instead, draw several samples and average:

$$\bar{\mathbf{X}} = \frac{1}{k}\sum_{t=1}^{k} \mathbf{X}_t,$$

where $\mathbf{X}_t$ are independent samples from the same distribution.

As k grows, the variance will decrease, as usual. Various Bernstein inequalities apply.

Matrix approximation by sampling

As an illustration of the theory, we cite a matrix-Bernstein result from J. Tropp (2015):

Theorem: Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. Construct a probability distribution for $\mathbf{X} \in \mathbb{R}^{m \times n}$ that satisfies

$$\mathbb{E}[\mathbf{X}] = \mathbf{A} \quad \text{and} \quad \|\mathbf{X}\| \leq R.$$

Define the per-sample second-moment: $v(\mathbf{X}) := \max\{\|\mathbb{E}[\mathbf{X}\mathbf{X}^*]\|, \|\mathbb{E}[\mathbf{X}^*\mathbf{X}]\|\}$.

Form the matrix sampling estimator: $\bar{\mathbf{X}}_k = \frac{1}{k}\sum_{t=1}^{k} \mathbf{X}_i$ where $\mathbf{X}_t \sim \mathbf{X}$ are iid.

Then $\mathbb{E}\|\bar{\mathbf{X}}_k - \mathbf{A}\| \leq \sqrt{\frac{2v(\mathbf{X})\log(m+n)}{k}} + \frac{2R\log(m+n)}{3k}$.

Furthermore, for all $s \geq 0$: $\mathbb{P}[\|\bar{\mathbf{X}}_k - \mathbf{A}\| \geq s] \leq (m+n)\exp\left(\frac{-ks^2/2}{v(\mathbf{X}) + 2Rs/3}\right)$.

Suppose that we want $\mathbb{E}\|\mathbf{A} - \bar{\mathbf{X}}\| \leq 2\epsilon$. The theorem says to pick

$$k \geq \max\left\{\frac{2v(\mathbf{X})\log(m+n)}{\epsilon^2}, \frac{2R\log(m+n)}{3\epsilon}\right\}$$

In other words, the number k of samples should be proportional to both $v(\mathbf{X})$ and to the upper bound R.

The scaling $k \sim \frac{1}{\epsilon^2}$ is discouraging, and unavoidable.

Matrix approximation by sampling: Matrix matrix multiplication

Given two matrices **B** and **C**, consider the task of evaluating

$$\begin{array}{ccc} \mathbf{A} & = & \mathbf{B} \quad \mathbf{C}. \\ m \times n & & m \times T \;\; T \times n \end{array}$$

Sampling approach:

1. Fix a probability distribution $\{p_t\}_{t=1}^{T}$ on the index vector $\{1, 2, \ldots, T\}$.
2. Draw a subset of k indices $J = \{t_1, t_2, \ldots, t_k\} \subseteq \{1, 2, \ldots, T\}$.
3. Use $\bar{\mathbf{A}} = \sum_{i=1}^{k} \frac{1}{p_{t_i}} \mathbf{B}(:, t_i)\mathbf{C}(t_i, :)$ to approximate **A**.

You get an unbiased estimator regardless of the probability distribution. But the computational profile depends critically how which one you choose. Common choices:

Uniform distribution: Very fast. Not very reliable or accurate.

Sample according to column/row norms: Cost is $O(mnk)$, which is much better than $O(mnT)$ when $k \ll T$. Better outcomes than uniform, but still not particularly good.

In either case, you need $k \sim \frac{1}{\epsilon^2}$ to attain precision ϵ.

Matrix approximation by sampling: Low rank approximation.

Given an $m \times n$ matrix $\mathbf{A}$, we seek a rank-k matrix $\bar{\mathbf{A}}$ such that $\|\mathbf{A} - \bar{\mathbf{A}}\|$ is small.

Sampling approach:

1. Draw vectors J and I holding k samples from the column and row indices, resp.
2. Form matrices $\mathbf{C}$ and $\mathbf{R}$ consisting of the corresponding columns and rows

$$\mathbf{C} = \mathbf{A}(:, J), \qquad \text{and} \qquad \mathbf{R} = \mathbf{A}(I, :).$$

3. Use as your approximation

$$\begin{array}{ccccc} \bar{\mathbf{A}} & = & \mathbf{C} & \mathbf{U} & \mathbf{R}, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

   where $\mathbf{U}$ is computed from information in $\mathbf{A}(I, J)$. (It should be an approximation to the optimal choice $\mathbf{U} = \mathbf{C}^{\dagger}\mathbf{A}\mathbf{R}^{\dagger}$.)

The computational profile depends crucially on the probability distribution that is used.

Uniform probabilities: Can be *very* cheap. But in general not reliable.

Probabilities from "leverage scores": Optimal distributions can be computed using the information in the top left and right singular vectors of $\mathbf{A}$. Then quite strong theorems can be proven on the quality of the approximation. Problem: Computing the probability distribution is as expensive as computing a partial SVD.

Wednesday: Randomized algorithms for linear algebraic computations

1. **Randomized low rank approximation:** "Randomized singular value decomposition" or "RSVD". Relatively well established material.
2. **Variations of algorithms for low rank approximation:** Single pass and streaming algorithms. Structured random embeddings. Matrix approximation via sampling.
3. **Samples of current research directions (time permitting):** Linear solvers. Least squares problems. Block Krylov methods. Rank structured matrices.

Friday: Randomized embeddings — theory and applications

4. **Randomized embeddings:** Reducing the effective dimension of point sets. Connections to Johnson-Lindenstrauss theory. Norm estimation.
5. **Analysis of the RSVD:** Outline of probabilistic error analysis for the RSVD. The relative merits of different classes of randomized embeddings.
6. **The column/row selection problem:** Interpolatory and CUR decompositions. Pivoting in QR and LU factorizations.

Recall the *randomized SVD (RSVD)* algorithm:

Objective: Given an $m \times n$ matrix $\mathbf{A}$, find an approximate rank-k partial SVD:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^* \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal. (We assume $k \ll \min(m, n)$.)

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix $\mathbf{G}$. `G = randn(n,k)`

A.2 Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{AG}$. `Y = A * G`

A.3 Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

B.1 Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`

B.2 Form the full SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`

B.3 Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

Observe that the final approximation is

$$\mathbf{A}_{\text{approx}} = \mathbf{U}\mathbf{D}\mathbf{V}^* = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

where the columns of $\mathbf{Q}$ form an ON basis for col($\mathbf{AG}$). In other words, $\mathbf{Q} = \texttt{orth}(\mathbf{AG})$.

Bound on the expectation of the error for Gaussian test matrices

Let $\mathbf{A}$ 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 such that $p \geq 2$.

Let $\mathbf{G}$ denote an $n \times (k+p)$ Gaussian matrix.

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

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

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

Notes:

- The bounds depend *only* on the singular values of $\mathbf{A}$.
- The spectral norm bound indicates substantial suboptimality in the algorithm in cases where the singular values decay slowly.
- Observe that the numbers of samples is only slightly larger than k. No "$1/\varepsilon^2$" term.

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

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

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

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

Proofs — Overview:

Let $\mathbf{A}$ 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 $\mathbf{Q}$ 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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\underset{k \times (k+p)}{\boldsymbol{\Omega}_1} = \underset{k \times n}{\mathbf{V}_1^*} \; \underset{n \times (k+p)}{\boldsymbol{\Omega}} \quad \text{and} \quad \underset{(n-k) \times (k+p)}{\boldsymbol{\Omega}_2} = \underset{(n-k) \times n}{\mathbf{V}_2^*} \; \underset{n \times (k+p)}{\boldsymbol{\Omega}}.$$

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

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

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

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

Recall: $\mathbf{A} = \mathbf{U}\begin{bmatrix}\mathbf{D}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{D}_2\end{bmatrix}\begin{bmatrix}\mathbf{V}_1^*\\ \mathbf{V}_2^*\end{bmatrix}$, $\begin{bmatrix}\Omega_1\\ \Omega_2\end{bmatrix} = \begin{bmatrix}\mathbf{V}_1^*\Omega\\ \mathbf{V}_2^*\Omega\end{bmatrix}$, $\mathbf{Y} = \mathbf{A}\Omega$, $\mathbf{P}$ proj$^{\mathrm{n}}$ onto Ran($\mathbf{Y}$).

Thm: Suppose $\mathbf{D}_1\Omega_1$ has full rank. Then $\|\mathbf{A} - \mathbf{PA}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\Omega_2\Omega_1^\dagger\|^2$.

Proof: The problem is rotationally invariant $\Rightarrow$ We can assume $\mathbf{U} = \mathbf{I}$ and so $\mathbf{A} = \mathbf{DV}^*$.

Simple calculation: $\|(\mathbf{I} - \mathbf{P})\mathbf{A}\|^2 = \|\mathbf{A}^*(\mathbf{I} - \mathbf{P})^2\mathbf{A}\| = \|\mathbf{D}(\mathbf{I} - \mathbf{P})\mathbf{D}\|$.

$$\mathrm{Ran}(\mathbf{Y}) = \mathrm{Ran}\left(\begin{bmatrix}\mathbf{D}_1\Omega_1\\ \mathbf{D}_2\Omega_2\end{bmatrix}\right) = \mathrm{Ran}\left(\begin{bmatrix}\mathbf{I}\\ \mathbf{D}_2\Omega_2\Omega_1^\dagger\mathbf{D}_1^{-1}\end{bmatrix}\mathbf{D}_1\Omega_1\right) = \mathrm{Ran}\left(\begin{bmatrix}\mathbf{I}\\ \mathbf{D}_2\Omega_2\Omega_1^\dagger\mathbf{D}_1^{-1}\end{bmatrix}\right)$$

Set $\mathbf{F} = \mathbf{D}_2\Omega_2\Omega_1^\dagger\mathbf{D}_1^{-1}$. Then $\mathbf{P} = \begin{bmatrix}\mathbf{I}\\ \mathbf{F}\end{bmatrix}(\mathbf{I} + \mathbf{F}^*\mathbf{F})^{-1}[\mathbf{I}\ \mathbf{F}^*]$. (Compare to $\mathbf{P}_{\mathrm{ideal}} = \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}$.)

Use properties of psd matrices: $\mathbf{I} - \mathbf{P} \preccurlyeq \cdots \preccurlyeq \begin{bmatrix}\mathbf{F}^*\mathbf{F} & -(\mathbf{I} + \mathbf{F}^*\mathbf{F})^{-1}\mathbf{F}^*\\ -\mathbf{F}(\mathbf{I} + \mathbf{F}^*\mathbf{F})^{-1} & \mathbf{I}\end{bmatrix}$

Conjugate by $\mathbf{D}$ to get $\mathbf{D}(\mathbf{I} - \mathbf{P})\mathbf{D} \preccurlyeq \begin{bmatrix}\mathbf{D}_1\mathbf{F}^*\mathbf{F}\mathbf{D}_1 & -\mathbf{D}_1(\mathbf{I} + \mathbf{F}^*\mathbf{F})^{-1}\mathbf{F}^*\mathbf{D}_2\\ -\mathbf{D}_2\mathbf{F}(\mathbf{I} + \mathbf{F}^*\mathbf{F})^{-1}\mathbf{D}_1 & \mathbf{D}_2^2\end{bmatrix}$

Diagonal dominance: $\|\mathbf{D}(\mathbf{I} - \mathbf{P})\mathbf{D}\| \leq \|\mathbf{D}_1\mathbf{F}^*\mathbf{F}\mathbf{D}_1\| + \|\mathbf{D}_2^2\| = \|\mathbf{D}_2\Omega_2\Omega_1^\dagger\|^2 + \|\mathbf{D}_2\|^2$.

Part 2 (out of 3) — bound on expectation of error when Ω is Gaussian:

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

Recall: $|||\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}|||^2 \leq |||\mathbf{D}_2|||^2 + |||\mathbf{D}_2\Omega_2\Omega_1^\dagger|||^2$, where $\Omega_1 = \mathbf{V}_1^*\,\Omega$ and $\Omega_2 = \mathbf{V}_2^*\,\Omega$.

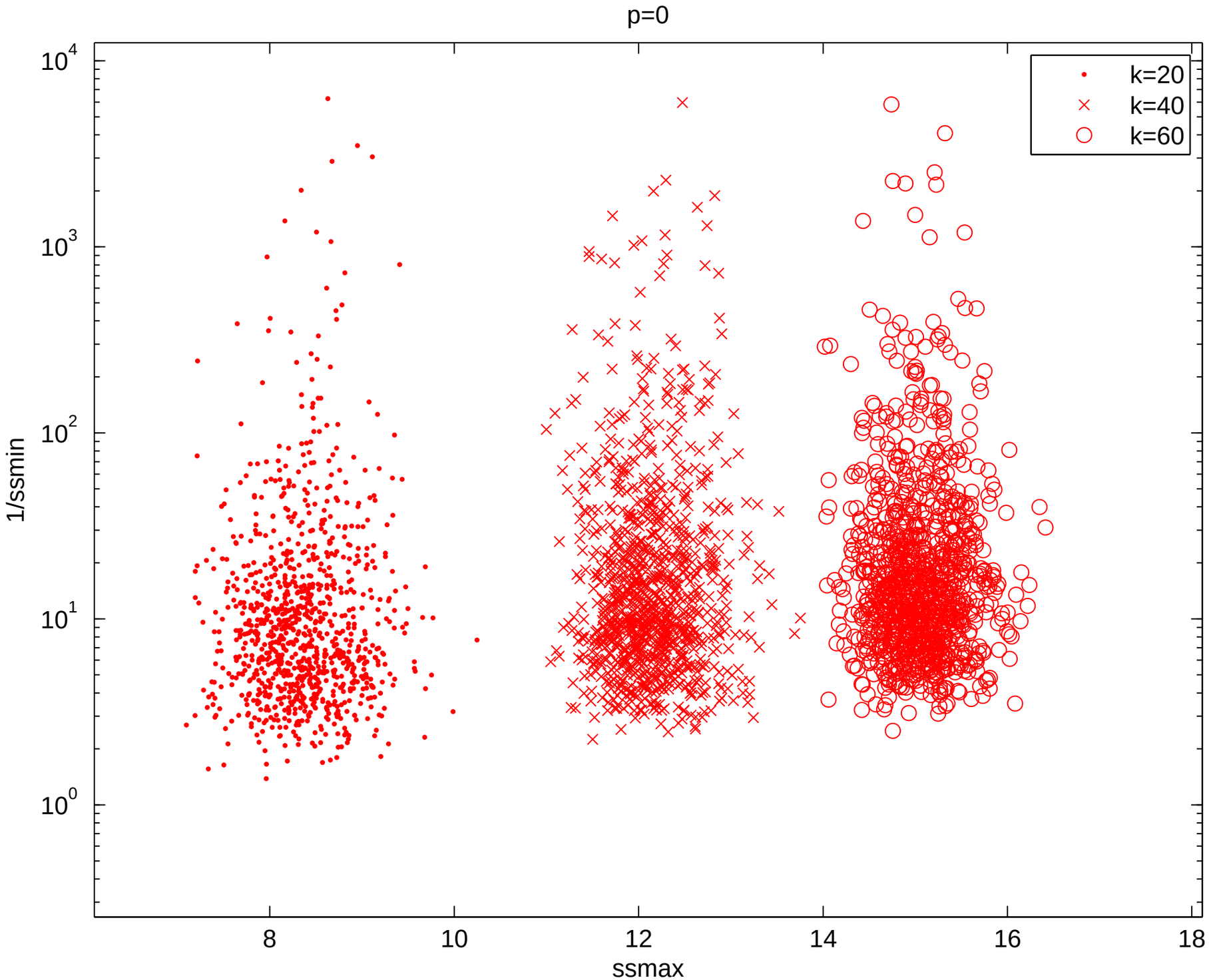
Assumption: Ω is drawn from a normal Gaussian distribution.

Since the Gaussian distribution is rotationally invariant, the matrices Ω_1 and Ω_2 also have a Gaussian distribution. (As a consequence, the matrices **U** and **V** do not enter the analysis and one could simply assume that **A** is diagonal, $\mathbf{A} = \mathrm{diag}(\sigma_1, \sigma_2, \dots)$.)

What is the distribution of $\Omega_1^\dagger$ when Ω_1 is a $k \times (k+p)$ Gaussian matrix?

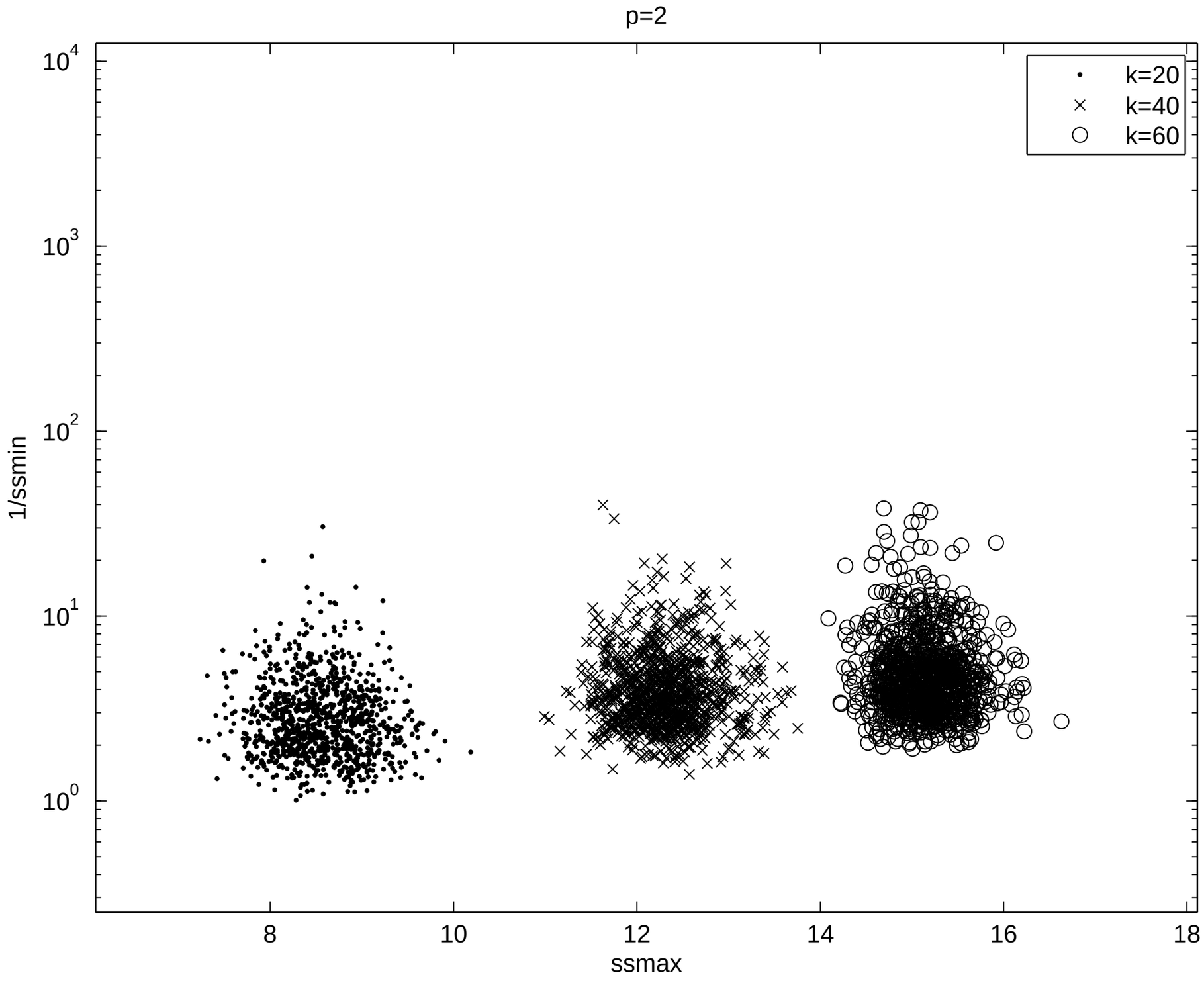
If $p = 0$, then $\|\Omega_1^\dagger\|$ is typically large, and is very unstable.

Scatter plot showing distribution of $1/\sigma_{\min}$ for $k \times (k+p)$ Gaussian matrices. $p = 0$



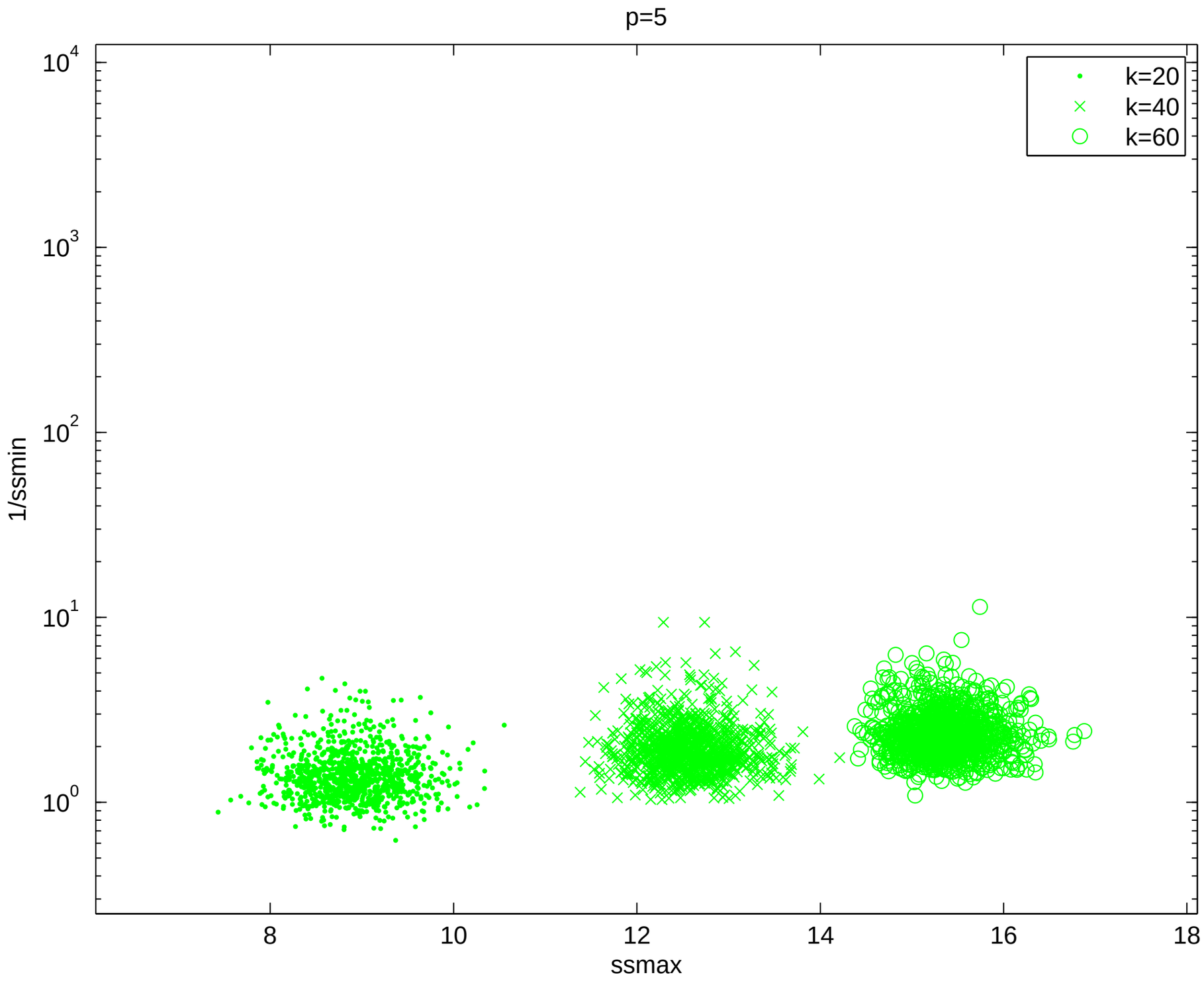
$1/\sigma_{\min}$ *is plotted against* $\sigma_{\max}$.

Scatter plot showing distribution of $1/\sigma_{\min}$ for $k \times (k+p)$ Gaussian matrices. $p = 2$



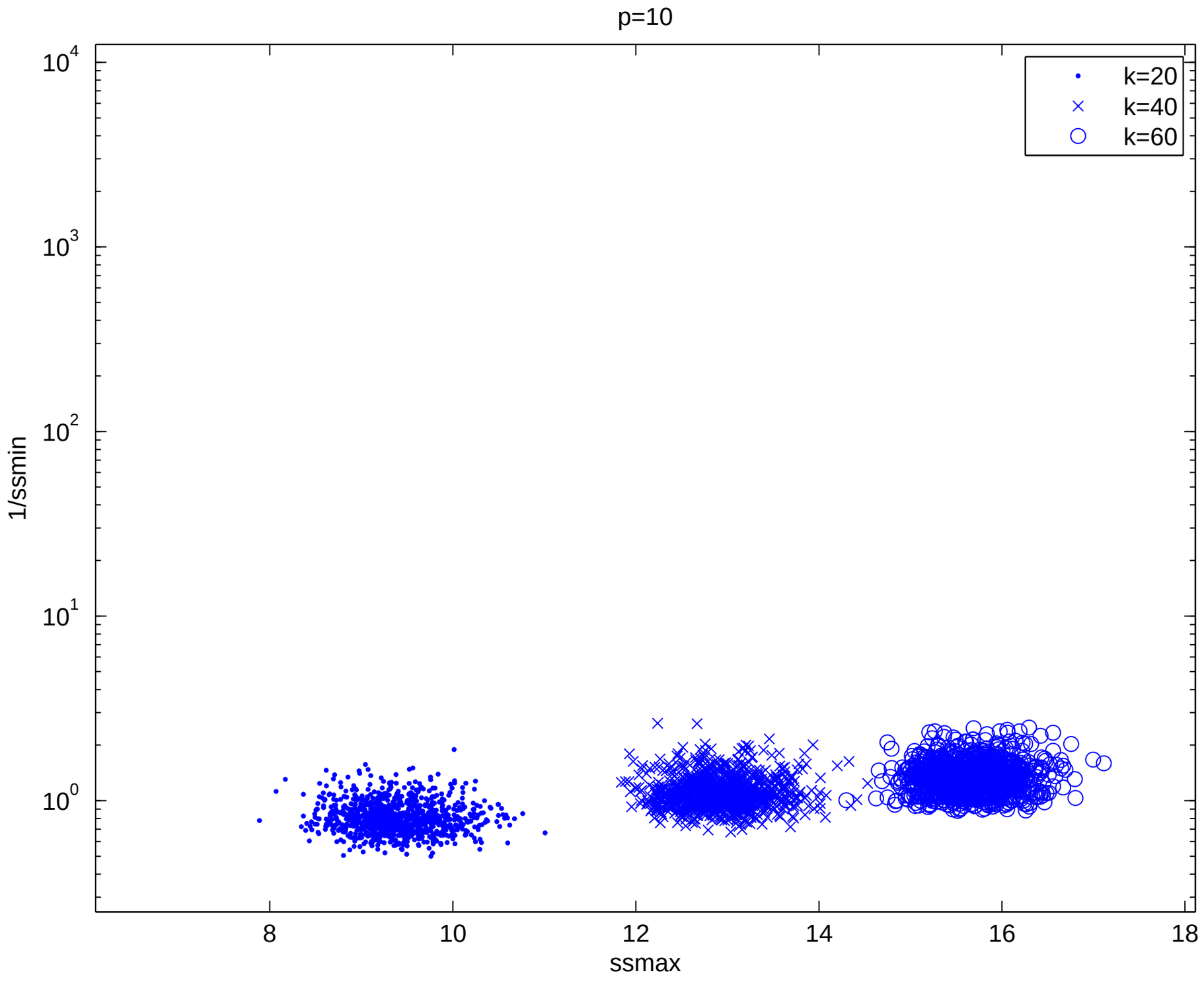
$1/\sigma_{\min}$ is plotted against $\sigma_{\max}$.

Scatter plot showing distribution of $1/\sigma_{\min}$ for $k \times (k+p)$ Gaussian matrices. $p = 5$



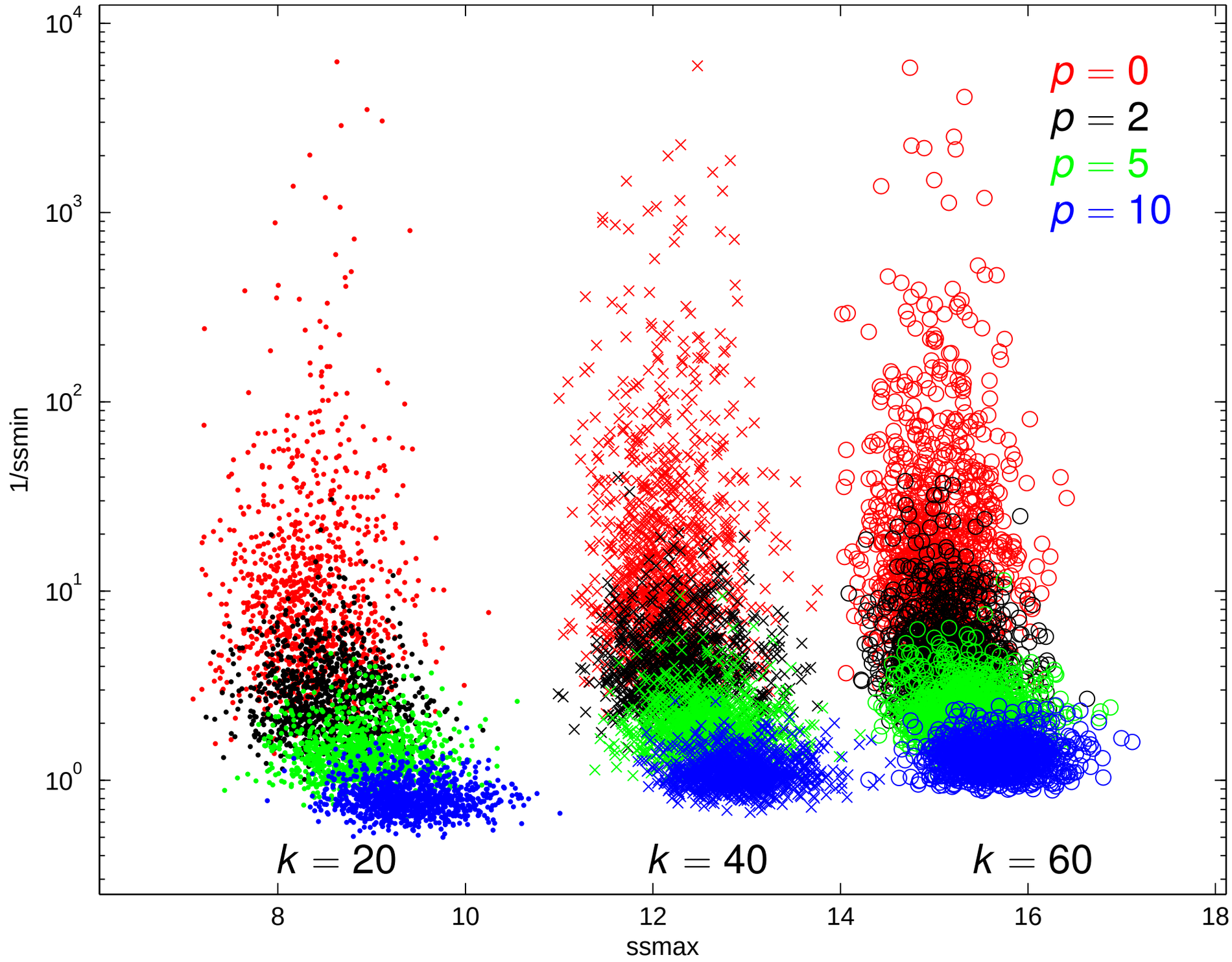
$1/\sigma_{\min}$ is plotted against $\sigma_{\max}$.

Scatter plot showing distribution of $1/\sigma_{\min}$ for $k \times (k+p)$ Gaussian matrices. $p = 10$



$1/\sigma_{\min}$ *is plotted against* $\sigma_{\max}$.

Scatter plot showing distribution of $k \times (k + p)$ Gaussian matrices.



$1/\sigma_{\min}$ *is plotted against* $\sigma_{\max}$.

Simplistic proof that a rectangular Gaussian matrix is well-conditioned:

Let **G** denote a $k \times \ell$ Gaussian matrix where $k < \ell$. Let "g" denote a generic $\mathcal{N}(0,1)$ variable and let "r_j" denote a generic random variable distributed like the square root of a χ^2_j variable. Then

$$\mathbf{G} \sim \begin{bmatrix} g & g & g & g & g & g & \cdots \\ g & g & g & g & g & g & \cdots \\ g & g & g & g & g & g & \cdots \\ g & g & g & g & g & g & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \end{bmatrix} \sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & 0 & \cdots \\ g & g & g & g & g & g & \cdots \\ g & g & g & g & g & g & \cdots \\ g & g & g & g & g & g & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \end{bmatrix}$$

$$\sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & 0 & \cdots \\ r_{k-1} & g & g & g & g & g & \cdots \\ 0 & g & g & g & g & g & \cdots \\ 0 & g & g & g & g & g & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \end{bmatrix} \sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & \cdots \\ r_{k-1} & r_{\ell-1} & 0 & 0 & 0 & \cdots \\ 0 & g & g & g & g & \cdots \\ 0 & g & g & g & g & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{bmatrix}$$

$$\sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & \cdots \\ r_{k-1} & r_{\ell-1} & 0 & 0 & 0 & \cdots \\ 0 & r_{k-2} & g & g & g & \cdots \\ 0 & 0 & g & g & g & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{bmatrix} \sim \cdots \sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & \cdots \\ r_{k-1} & r_{\ell-1} & 0 & 0 & 0 & \cdots \\ 0 & r_{k-2} & r_{\ell-2} & 0 & 0 & \cdots \\ 0 & 0 & r_{k-3} & r_{\ell-3} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{bmatrix}$$

Gershgorin's circle theorem will now show that **G** is highly likely to be well-conditioned if, e.g., $\ell = 2k$. More sophisticated methods are required to get to $\ell = k + 2$.

Some results on Gaussian matrices. *Adapted from HMT 2009/2011; Gordon (1985,1988) for Proposition 1; Chen & Dongarra (2005) for Propositions 2 and 4; Bogdanov (1998) for Proposition 3.*

Proposition 1: Let $\mathbf{G}$ be a Gaussian matrix. Then

$$\begin{aligned}\left(\mathbb{E}\left[\|\mathbf{SGT}\|_{\mathrm{F}}^2\right]\right)^{1/2} &\leq \|\mathbf{S}\|_{\mathrm{F}}\,\|\mathbf{T}\|_{\mathrm{F}} \\ \mathbb{E}\left[\|\mathbf{SGT}\|\right] &\leq \|\mathbf{S}\|\,\|\mathbf{T}\|_{\mathrm{F}} + \|\mathbf{S}\|_{\mathrm{F}}\,\|\mathbf{T}\|\end{aligned}$$

Proposition 2: Let $\mathbf{G}$ be a Gaussian matrix of size $k \times k+p$ where $p \geq 2$. Then

$$\begin{aligned}\left(\mathbb{E}\left[\|\mathbf{G}^{\dagger}\|_{\mathrm{F}}^2\right]\right)^{1/2} &\leq \sqrt{\frac{k}{p-1}} \\ \mathbb{E}\left[\|\mathbf{G}^{\dagger}\|\right] &\leq \frac{e\sqrt{k+p}}{p}.\end{aligned}$$

Proposition 3: Suppose h is Lipschitz $|h(\mathbf{X}) - h(\mathbf{Y})| \leq L\|\mathbf{X}-\mathbf{Y}\|_{\mathrm{F}}$ and $\mathbf{G}$ is Gaussian. Then

$$\mathbb{P}\left[h(\mathbf{G}) > \mathbb{E}[h(\mathbf{G})] + L\,u\right] \leq e^{-u^2/2}.$$

Proposition 4: Suppose $\mathbf{G}$ is Gaussian of size $k \times k+p$ with $p \geq 4$. Then for $t \geq 1$:

$$\begin{aligned}\mathbb{P}\left[\|\mathbf{G}^{\dagger}\|_{\mathrm{F}} \geq \sqrt{\frac{3k}{p+1}}t\right] &\leq t^{-p} \\ \mathbb{P}\left[\|\mathbf{G}^{\dagger}\| \geq \frac{e\sqrt{k+p}}{p+1}t\right] &\leq t^{-(p+1)}\end{aligned}$$

Recall: $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\Omega_2\Omega_1^\dagger\|^2$, where Ω_1 and Ω_2 are Gaussian and Ω_1 is $k \times k + p$.

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

Proof: First observe that

$$\mathbb{E}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| = \mathbb{E}\big(\|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\Omega_2\Omega_1^\dagger\|^2\big)^{1/2} \leq \|\mathbf{D}_2\| + \mathbb{E}\|\mathbf{D}_2\Omega_2\Omega_1^\dagger\|.$$

Condition on Ω_1 and use Proposition 1:

$$\begin{aligned}\mathbb{E}\|\mathbf{D}_2\Omega_2\Omega_1^\dagger\| &\leq \mathbb{E}\big[\|\mathbf{D}_2\|\,\|\Omega_1^\dagger\|_{\mathrm{F}} + \|\mathbf{D}_2\|_{\mathrm{F}}\,\|\Omega_1^\dagger\|\big] \\ &\leq \{\text{Hölder}\} \leq \|\mathbf{D}_2\|\left(\mathbb{E}\|\Omega_1^\dagger\|_{\mathrm{F}}^2\right)^{1/2} + \|\mathbf{D}_2\|_{\mathrm{F}}\,\mathbb{E}\|\Omega_1^\dagger\|.\end{aligned}$$

Proposition 2 now provides bounds for $\mathbb{E}\|\Omega_1^\dagger\|_{\mathrm{F}}^2$ and $\mathbb{E}\|\Omega_1^\dagger\|$ and we get

$$\mathbb{E}\|\mathbf{D}_2\Omega_2\Omega_1^\dagger\| \leq \sqrt{\frac{k}{p-1}}\|\mathbf{D}_2\| + \frac{e\sqrt{k+p}}{p}\|\mathbf{D}_2\|_{\mathrm{F}} = \sqrt{\frac{k}{p-1}}\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j>k}\sigma_j^2\right)^{1/2}.$$

Some results on Gaussian matrices. *Adapted from HMT2009/2011; Gordon (1985,1988) for Proposition 1; Chen & Dongarra (2005) for Propositions 2 and 4; Bogdanov (1998) for Proposition 3.*

Proposition 1: Let $\mathbf{G}$ be a Gaussian matrix. Then

$$\begin{aligned}\left(\mathbb{E}\left[\|\mathbf{SGT}\|_{\mathrm{F}}^2\right]\right)^{1/2} &\leq \|\mathbf{S}\|_{\mathrm{F}}\,\|\mathbf{T}\|_{\mathrm{F}} \\ \mathbb{E}\left[\|\mathbf{SGT}\|\right] &\leq \|\mathbf{S}\|\,\|\mathbf{T}\|_{\mathrm{F}} + \|\mathbf{S}\|_{\mathrm{F}}\,\|\mathbf{T}\|\end{aligned}$$

Proposition 2: Let $\mathbf{G}$ be a Gaussian matrix of size $k \times k+p$ where $p \geq 2$. Then

$$\begin{aligned}\left(\mathbb{E}\left[\|\mathbf{G}^\dagger\|_{\mathrm{F}}^2\right]\right)^{1/2} &\leq \sqrt{\frac{k}{p-1}} \\ \mathbb{E}\left[\|\mathbf{G}^\dagger\|\right] &\leq \frac{e\sqrt{k+p}}{p}.\end{aligned}$$

Proposition 3: Suppose h is Lipschitz $|h(\mathbf{X}) - h(\mathbf{Y})| \leq L\|\mathbf{X}-\mathbf{Y}\|_{\mathrm{F}}$ and $\mathbf{G}$ is Gaussian. Then

$$\mathbb{P}\left[h(\mathbf{G}) > \mathbb{E}[h(\mathbf{G})] + L\,u\right] \leq e^{-u^2/2}.$$

Proposition 4: Suppose $\mathbf{G}$ is Gaussian of size $k \times k+p$ with $p \geq 4$. Then for $t \geq 1$:

$$\begin{aligned}\mathbb{P}\left[\|\mathbf{G}^\dagger\|_{\mathrm{F}} \geq \sqrt{\frac{3k}{p+1}}t\right] &\leq t^{-p} \\ \mathbb{P}\left[\|\mathbf{G}^\dagger\| \geq \frac{e\sqrt{k+p}}{p+1}t\right] &\leq t^{-(p+1)}\end{aligned}$$

Recall: $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\Omega_2\Omega_1^\dagger\|^2$, where Ω_1 and Ω_2 are Gaussian and Ω_1 is $k \times k+p$.

Theorem: With probability at least $1 - 2\,t^{-p} - e^{-u^2/2}$ it holds that

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

Proof: Set $E_t = \left\{\|\Omega_1\| \leq \frac{e\sqrt{k+p}}{p+1}t \quad \text{and} \quad \|\Omega_1^\dagger\|_{\mathrm{F}} \leq \sqrt{\frac{3k}{p+1}}\,t\right\}$. By Proposition 4: $\mathbb{P}(E_t^{\mathrm{c}}) \leq 2\,t^{-p}$.

Set $h(\mathbf{X}) = \|\mathbf{D}_2\mathbf{X}\Omega_1^\dagger\|$. A direct calculation shows

$$|h(\mathbf{X}) - h(\mathbf{Y})| \leq \|\mathbf{D}_2\|\,\|\Omega_1^\dagger\|\,\|\mathbf{X} - \mathbf{y}\|_{\mathrm{F}}.$$

Hold Ω_1 fixed and take the expectation on Ω_2. Then Proposition 1 applies and so

$$\mathbb{E}\big[h(\Omega_2) \;\big|\; \Omega_1\big] \leq \|\mathbf{D}_2\|\,\|\Omega_1^\dagger\|_{\mathrm{F}} + \|\mathbf{D}_2\|_{\mathrm{F}}\,\|\Omega_1^\dagger\|.$$

Now use Proposition 3 (concentration of measure)

$$\mathbb{P}\Big[\underbrace{\|\mathbf{D}_2\Omega_2\Omega_1^\dagger\|}_{=h(\Omega_2)} > \underbrace{\|\mathbf{D}_2\|\,\|\Omega_1^\dagger\|_{\mathrm{F}} + \|\mathbf{D}_2\|_{\mathrm{F}}\,\|\Omega_1^\dagger\|}_{=\mathbb{E}[h(\Omega_2)]} + \underbrace{\|\mathbf{D}_2\|\,\|\Omega_1^\dagger\|}_{=L}\,u \;\Big|\; E_t\Big] < e^{-u^2/2}.$$

When E_t holds true, we have bounds on the "badness" of $\Omega_1^\dagger$:

$$\mathbb{P}\left[\|\mathbf{D}_2\Omega_2\Omega_1^\dagger\| > \|\mathbf{D}_2\|\sqrt{\frac{3k}{p+1}}t + \|\mathbf{D}_2\|_{\mathrm{F}}\,\frac{e\sqrt{k+p}}{p+1}t + \|\mathbf{D}_2\|\,\frac{e\sqrt{k+p}}{p+1}\,ut \;\middle|\; E_t\right] < e^{-u^2/2}.$$

The theorem is obtained by using $\mathbb{P}(E_t^{\mathrm{c}}) \leq 2\,t^{-p}$ to remove the conditioning of E_t.

Power method for improving accuracy:

The error depends on how quickly the singular values decay.

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 its singular values are

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

Much faster decay — so let us use the sample matrix

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

instead of

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

Input: An $m \times n$ matrix **A**, a target rank ℓ, and a small integer q.	
Output: Rank-ℓ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times \ell$ Gaussian matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times \ell$ sample matrix $\mathbf{Y} = (\mathbf{A}\,\mathbf{A}^*)^q\,\mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(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, with $\mathbf{A}^{\text{computed}} = \mathbf{UDV}^*$, the expectation of the error satisfies:

$$\mathbb{E}\left[\|\mathbf{A} - \mathbf{A}^{\text{computed}}\|\right] \leq \left(1 + 4\sqrt{\frac{2\,\min(m,n)}{k-1}}\right)^{1/(2q+1)} \sigma_{k+1}(\mathbf{A}). \tag{5}$$

  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 (5) 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.

Structured randomized embeddings "Fast J-L transforms"

Recall that the RSVD can be accelerated by using a random distribution of test matrices that can be applied more rapidly to vectors than Gaussian matrices.
Some options we mentioned that improve on the $O(mnk)$ cost of Gaussian matrices:

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn\log(k))$.
- Chains of Given's rotations ("Kac's random walk"). Cost is $O(mn\log(k))$.
- "Sparse sign matrix". Place r random entries in each row of Ω. (Say $r\{2, 3, 4, 5\}$.) Cost is now $O(mn)$!

Question: Which type of random matrix should I use for the sketching?

Structured randomized embeddings "Fast J-L transforms"

Recall that the RSVD can be accelerated by using a random distribution of test matrices that can be applied more rapidly to vectors than Gaussian matrices.
Some options we mentioned that improve on the $O(mnk)$ cost of Gaussian matrices:

- Randomized trigonometric transforms (FFT, Hadamard, etc). Cost is $O(mn\log(k))$.
- Chains of Given's rotations ("Kac's random walk"). Cost is $O(mn\log(k))$.
- "Sparse sign matrix". Place r random entries in each row of Ω. (Say $r\{2, 3, 4, 5\}$.) Cost is now $O(mn)$!

Question: Which type of random matrix should I use for the sketching?

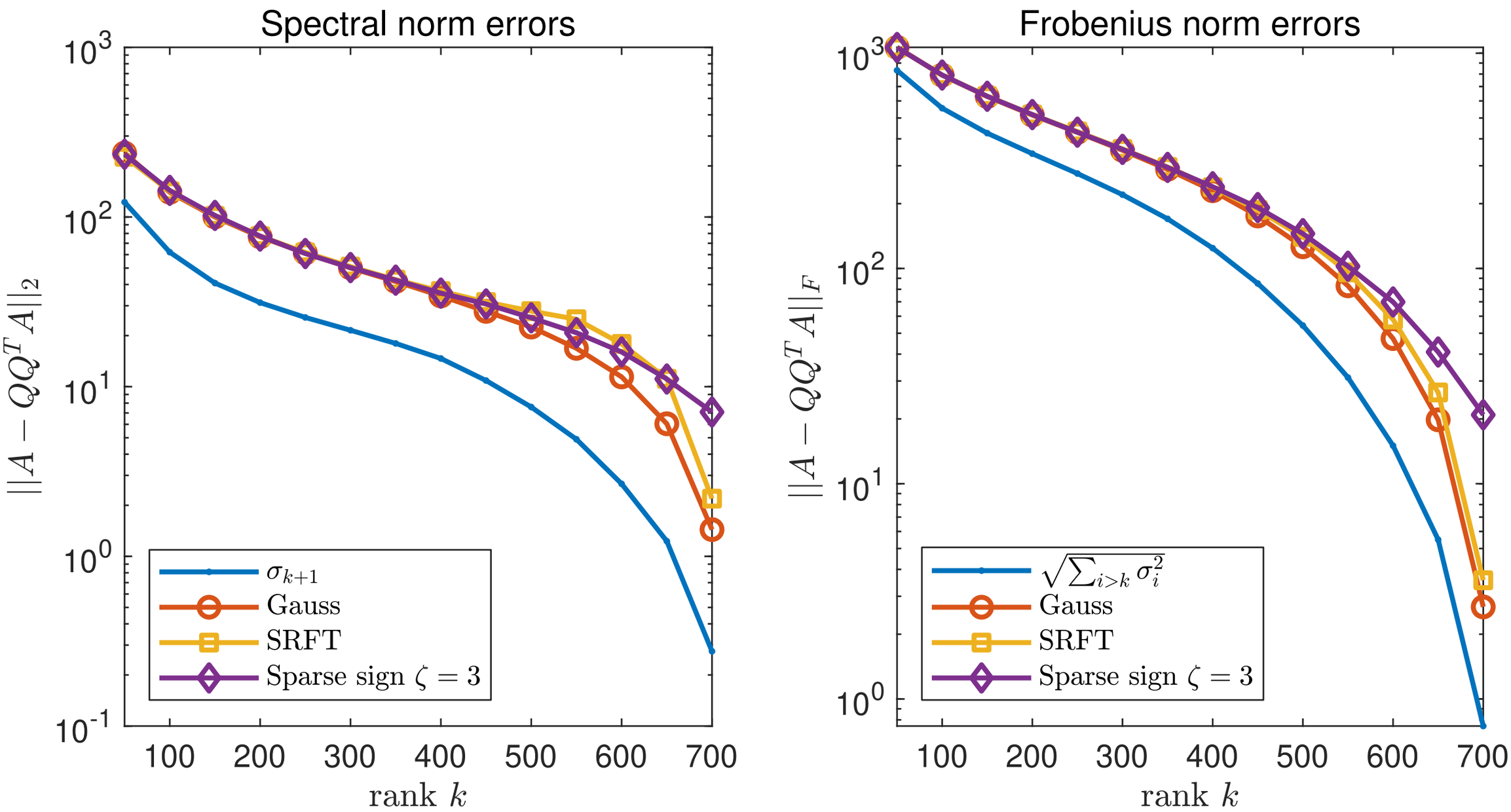
The *theory* for structured maps is quite weak — it appears to indicate that enormous over-sampling is required.

Instead of invoking theory, we will attempt to answer the question via *numerical experiments*. We will compare:

- Optimality: How good of a basis for the column space do you get?
- Computational cost: What is the *practical* speed?

Comparison of different random matrices — accuracy

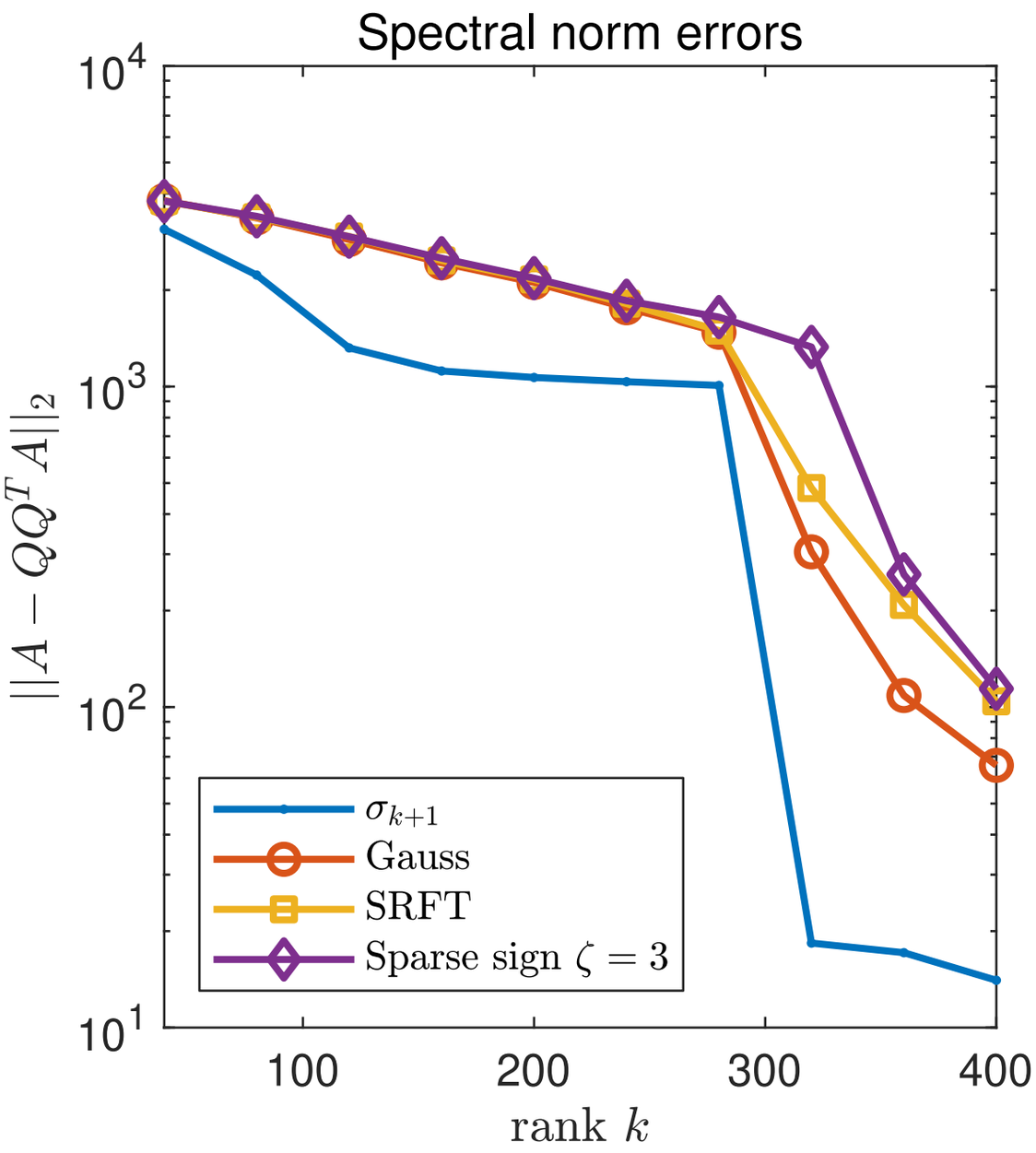
Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

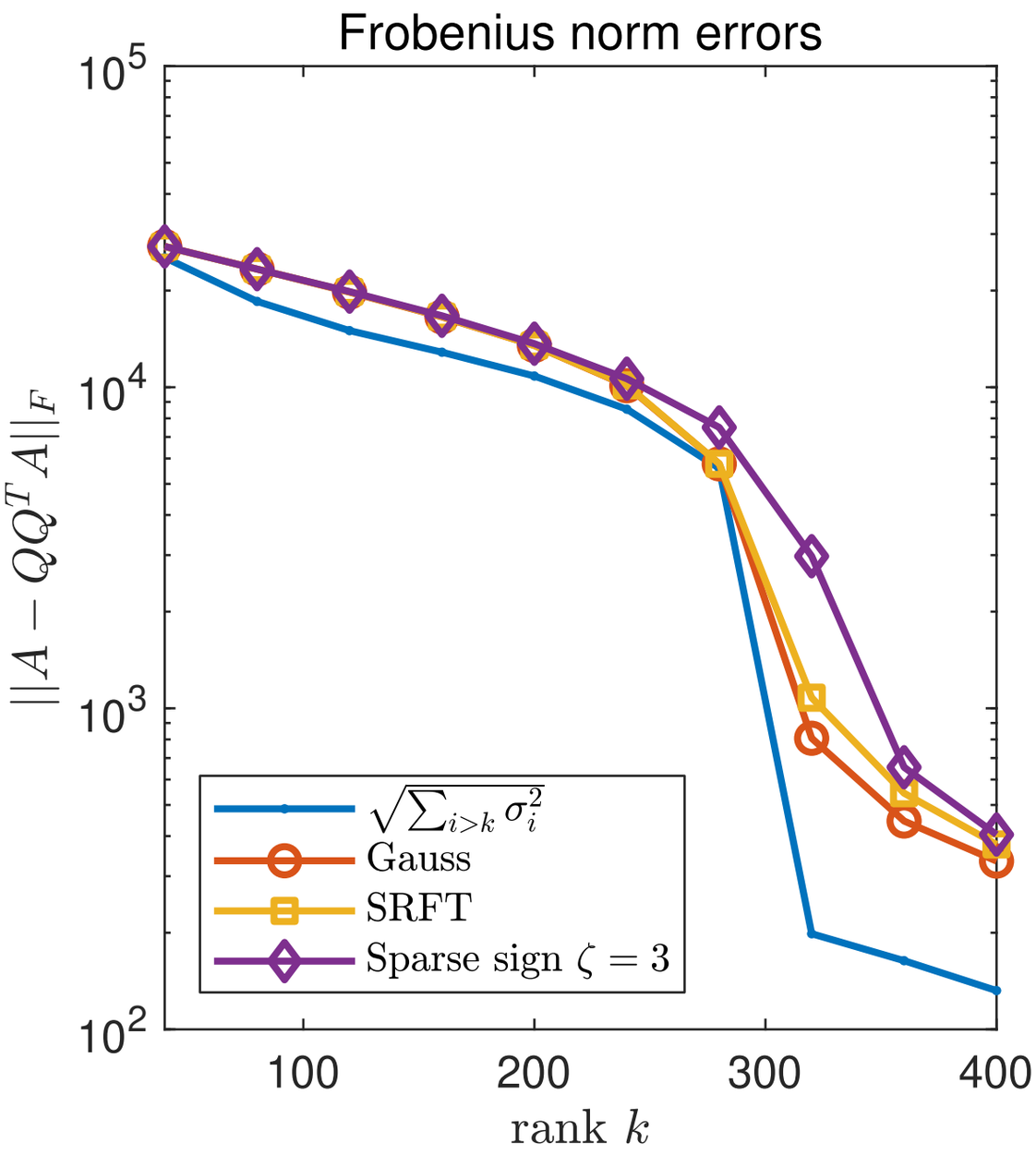


*The "*MNIST*" test matrix is dense and of size* 784 $\times$ 60 000 *where each column holds one hard drawn digit between 0 and 9. The matrix is* 80% *sparse.*

Comparison of different random matrices — accuracy

Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

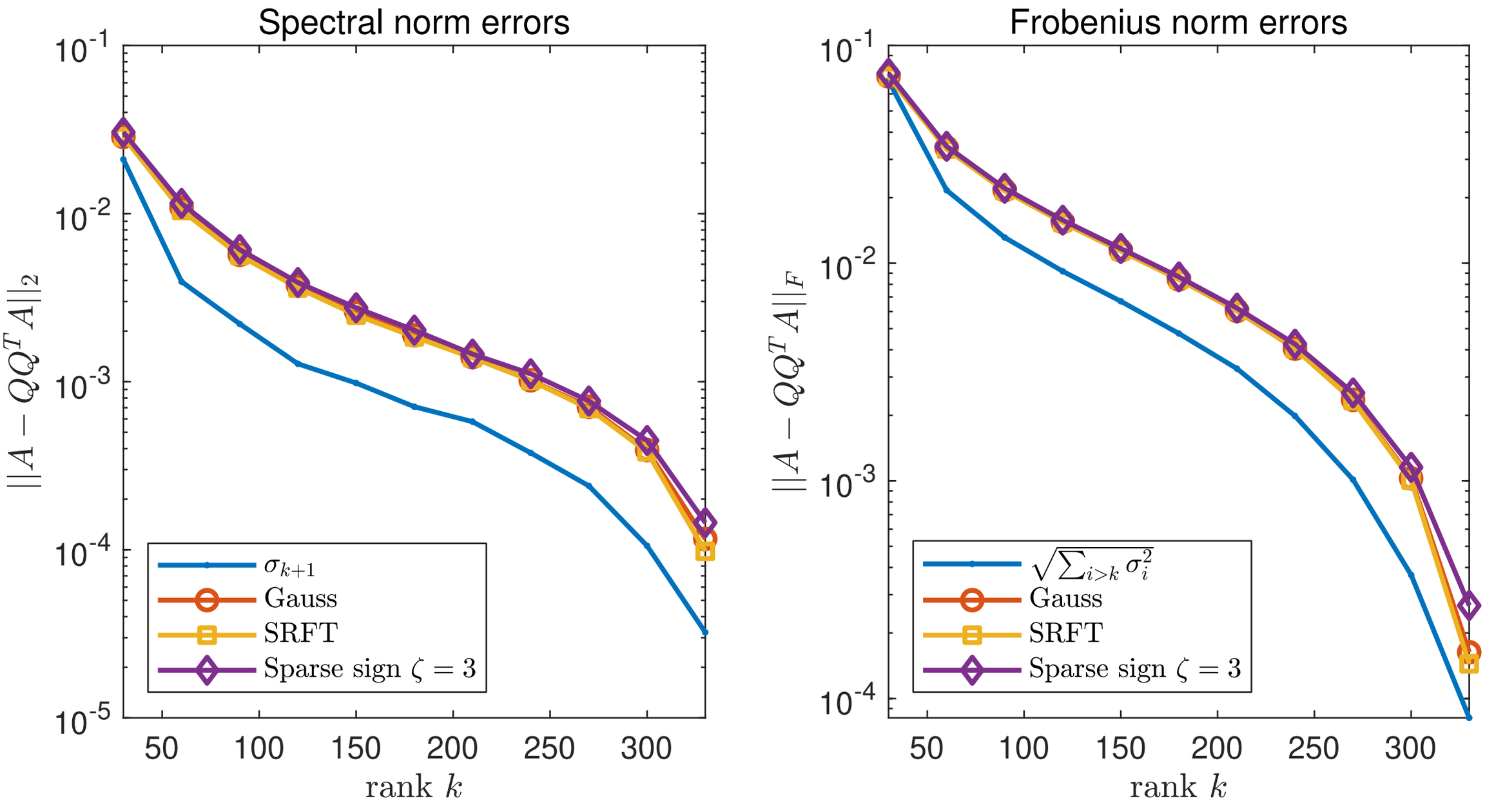




*The "*LARGE*" test matrix is taken from a linear programming example. It is sparse, of size* 4 282 × 8 617, *with* 20 635 *nonzero entries.*

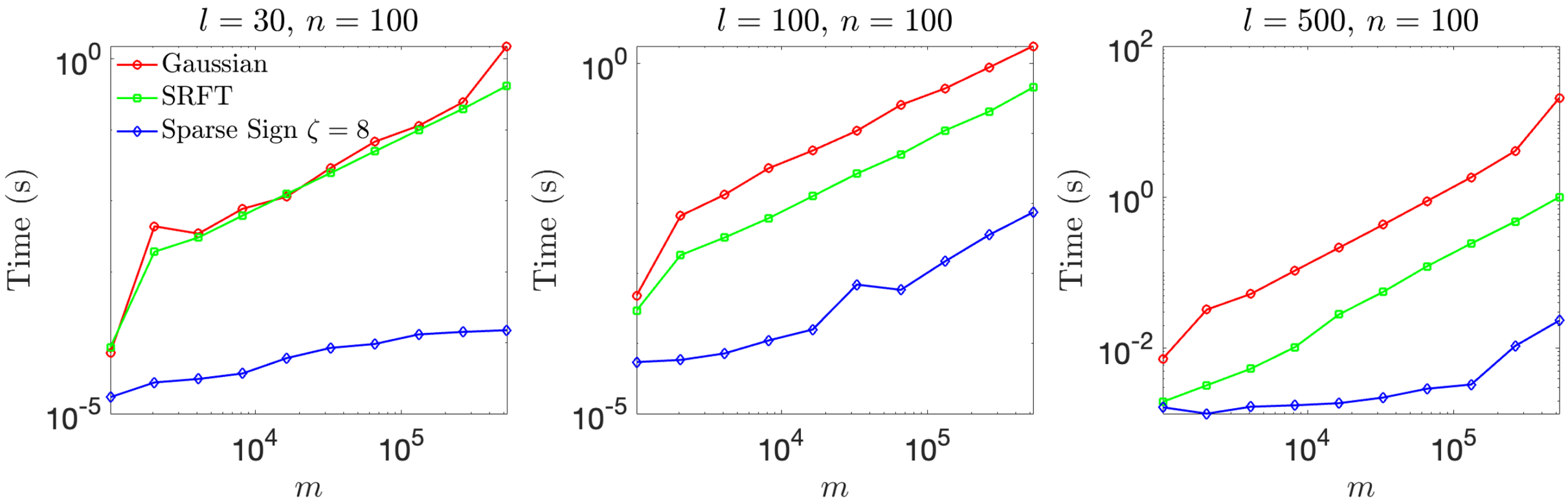
Comparison of different random matrices — accuracy

Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.



The "SNN" test matrix has been used in the CUR literature before. It is an artificial sparse matrix of size $1\,000 \times 1\,000$.

Comparison of different random matrices — execution time



The runtime of applying different subspace embeddings $\mathbf{\Omega} \in \mathbb{R}^{\ell \times m}$ *to an arbitrary dense matrix of size* $m \times n$*, scaled with respect to the ambient dimension* m*, at different embedding dimension* l *and a fixed number of columns* $n = 100$*.*
(Note: This n *is artificially small, but the scaling with* n *is linear.)*

Note: Observe that the dimension of the sketch is quite high in these examples.

Take-aways from numerical experiments:

- Gaussian matrices are highly recommended. Excellent general purpose tools.
- "Sparse random" is *very fast* in all environments. Slightly less accurate.
- Subsampled trigonometric transforms are about as accurate as Gaussians. When the rank is large (say 500 or 1000), you see substantial speed gain.

Wednesday: Randomized algorithms for linear algebraic computations

1. **Randomized low rank approximation:** "Randomized singular value decomposition" or "RSVD". Relatively well established material.
2. **Variations of algorithms for low rank approximation:** Single pass and streaming algorithms. Structured random embeddings. Matrix approximation via sampling.
3. **Samples of current research directions (time permitting):** Linear solvers. Least squares problems. Block Krylov methods. Rank structured matrices.

Friday: Randomized embeddings — theory and applications

4. **Randomized embeddings:** Reducing the effective dimension of point sets. Connections to Johnson-Lindenstrauss theory. Norm estimation.
5. **Analysis of the RSVD:** Outline of probabilistic error analysis for the RSVD. The relative merits of different classes of randomized embeddings.
6. **The column/row selection problem:** Interpolatory and CUR decompositions. Pivoting in QR and LU factorizations.

Finding spanning rows and columns — CUR and interpolatory decompositions

In some applications, finding sets of columns/rows that span the column/row space is the primary task. To illustrate, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a rank $k < \min(m, n)$, and seek to compute a *column interpolatory decomposition (ID)*

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{C}} \quad \underset{k \times n}{\mathbf{Z}},$$

where $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$ holds a subset of k columns of $\mathbf{A}$.

Finding spanning rows and columns — CUR and interpolatory decompositions

In some applications, finding sets of columns/rows that span the column/row space is the primary task. To illustrate, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a rank $k < \min(m, n)$, and seek to compute a *column interpolatory decomposition (ID)*

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{C}} \quad \underset{k \times n}{\mathbf{Z}},$$

where $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$ holds a subset of k columns of $\mathbf{A}$.

Why insist on using the columns as a basis?

- The index vector J_{S} is useful in data interpretation.
- If $\mathbf{A}$ is sparse, then $\mathbf{C}$ is sparse.
- If $\mathbf{A}$ is non-negative, then $\mathbf{C}$ is non-negative.
- Storage efficient in that $\mathbf{C}$ often does not need to be formed.
- Can be faster to compute than randomized SVD, and other competitors.

Finding spanning rows and columns — CUR and interpolatory decompositions

In some applications, finding sets of columns/rows that span the column/row space is the primary task. To illustrate, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a rank $k < \min(m, n)$, and seek to compute a *column interpolatory decomposition (ID)*

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{C}} \quad \underset{k \times n}{\mathbf{Z}},$$

where $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$ holds a subset of k columns of $\mathbf{A}$.

Variations: You can of course use rows to span the row space instead / as well:

- The *row interpolatory decomposition* takes the form

  $$\mathbf{A} \approx \mathbf{X}\mathbf{R}$$

  where $\mathbf{R} = \mathbf{A}(I_{\mathrm{S}}, :)$ holds a subset of k rows of $\mathbf{A}$.

- The *two sided interpolatory decomposition* takes the form

  $$\mathbf{A} \approx \mathbf{X}\mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}})\mathbf{Z}.$$

- The *CUR decomposition* takes the form

  $$\mathbf{A} \approx \mathbf{C}\mathbf{U}\mathbf{R},$$

  where $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$, where $\mathbf{R} = \mathbf{A}(I_{\mathrm{S}}, :)$, and where $\mathbf{U}$ is some $k \times k$ matrix.

Finding spanning rows and columns — CUR and interpolatory decompositions

In some applications, finding sets of columns/rows that span the column/row space is the primary task. To illustrate, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a rank $k < \min(m, n)$, and seek to compute a *column interpolatory decomposition (ID)*

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{C}} \quad \underset{k \times n}{\mathbf{Z}},$$

where $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$ holds a subset of k columns of $\mathbf{A}$.

Traditional algorithms:

- Column pivoted QR ("Gram Schmidt").
- Fully pivoted LU (closely related to "cross approximation").

Cost is $O(mnk)$, which is good. But communication intensive $\rightarrow$ large prefactors.

Can in theory be far from optimal, but tends to work well in practice.

(Provably close to optimal methods do exist — e.g. Gu-Eisenstat.)

Note: Faster factorizations (unpivoted QR, partially pivoted LU,...) do not work.

Finding spanning rows and columns — CUR and interpolatory decompositions

In some applications, finding sets of columns/rows that span the column/row space is the primary task. To illustrate, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a rank $k < \min(m, n)$, and seek to compute a *column interpolatory decomposition (ID)*

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{C}} \underset{k \times n}{\mathbf{Z}},$$

where $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$ holds a subset of k columns of $\mathbf{A}$.

Randomized sampling:

The idea is to draw an index vector J_{S} by through random sampling from $\{1, 2, \ldots, n\}$.

A uniform distribution sometimes works well. More typically, for this to be reliable, you need to rely on so called *leverage scores*. These are expensive to compute accurately, but can sometimes be approximated at low cost.

The number of samples required is typically substantially larger than the minimal number. A two-step approach where you first sample generously, and then pick the best columns among the sample can sometimes work.

Attractive for huge matrices where you cannot afford to form the entire matrix.

Finding spanning rows and columns — CUR and interpolatory decompositions

In some applications, finding sets of columns/rows that span the column/row space is the primary task. To illustrate, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a rank $k < \min(m, n)$, and seek to compute a *column interpolatory decomposition (ID)*

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{C}} \quad \underset{k \times n}{\mathbf{Z}},$$

where $\mathbf{C} = \mathbf{A}(:, J_s)$ holds a subset of k columns of $\mathbf{A}$.

Random embedding + classical pivoting:

(1) Apply a random embedding $\boldsymbol{\Omega}$ to the columns of $\mathbf{A}$.

(2) Execute a classical pivoting method on $\boldsymbol{\Omega}\mathbf{A}$.

Finding spanning rows and columns — CUR and interpolatory decompositions

In some applications, finding sets of columns/rows that span the column/row space is the primary task. To illustrate, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, a rank $k < \min(m, n)$, and seek to compute a *column interpolatory decomposition (ID)*

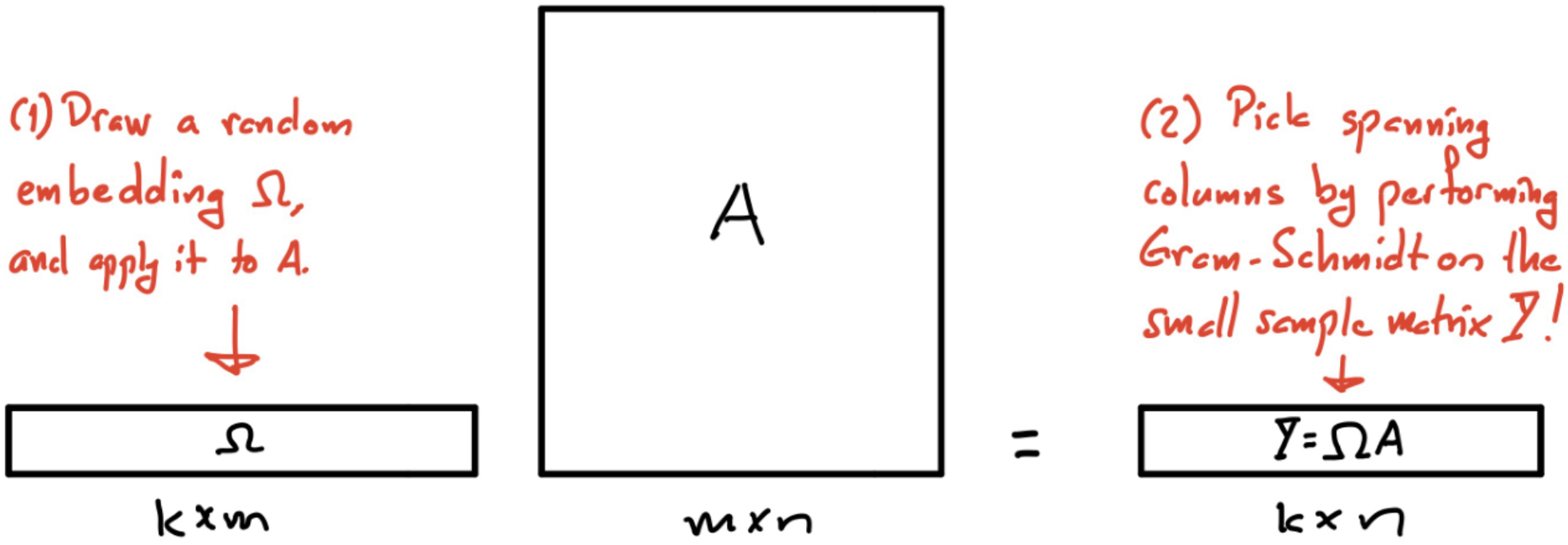
$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{C}} \underset{k \times n}{\mathbf{Z}},$$

where $\mathbf{C} = \mathbf{A}(:, J_S)$ holds a subset of k columns of $\mathbf{A}$.

Random embedding + classical pivoting:

(1) Apply a random embedding $\boldsymbol{\Omega}$ to the columns of $\mathbf{A}$.

(2) Execute a classical pivoting method on $\boldsymbol{\Omega}\mathbf{A}$.



The column selection problem — through a *sketch*

Simple theorem: Let **A** be an $m \times n$ matrix of exact rank k. Suppose:

(1) We have by some means computed a factorization

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}}.$$

(2) We have solved the column selection problem for **F**, so that

$$\underset{k \times n}{\mathbf{F}} = \underset{k \times k}{\mathbf{F}(:, J_{\mathrm{S}})} \; \underset{k \times n}{\mathbf{Z}}.$$

Then, *automatically,* we have also solved the column selection problem for **A**:

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{A}(:, J_{\mathrm{S}})} \; \underset{k \times n}{\mathbf{Z}}.$$

The column selection problem — through a *sketch*

Simple theorem: Let **A** be an $m \times n$ matrix of exact rank k. Suppose:

(1) We have by some means computed a factorization

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}}.$$

(2) We have solved the column selection problem for **F**, so that

$$\underset{k \times n}{\mathbf{F}} = \underset{k \times k}{\mathbf{F}(:, J_{\mathrm{S}})} \; \underset{k \times n}{\mathbf{Z}}.$$

Then, *automatically,* we have also solved the column selection problem for **A**:

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{A}(:, J_{\mathrm{S}})} \; \underset{k \times n}{\mathbf{Z}}.$$

Proof: Assume that

(*) $$\mathbf{A} = \mathbf{EF}$$

and that

(**) $$\mathbf{F} = \mathbf{F}(:, J_{\mathrm{S}})\mathbf{Z}.$$

Then

$$\mathbf{A}(:, J_{\mathrm{S}})\mathbf{Z} \overset{(*)}{=} \mathbf{EF}(:, J_{\mathrm{S}})\mathbf{Z} \overset{(**)}{=} \mathbf{EF} = \mathbf{A}.$$

The column selection problem — through a *sketch*

Simple theorem: Let **A** be an $m \times n$ matrix of exact rank k. Suppose:

(1) We have by some means computed a factorization

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}}.$$

(2) We have solved the column selection problem for **F**, so that

$$\underset{k \times n}{\mathbf{F}} = \underset{k \times k}{\mathbf{F}(:, J_{\mathrm{S}})} \; \underset{k \times n}{\mathbf{Z}}.$$

Then, *automatically,* we have also solved the column selection problem for **A**:

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{A}(:, J_{\mathrm{S}})} \; \underset{k \times n}{\mathbf{Z}}.$$

Randomized embedding: Ideal way to find the matrix **F**!

Draw a $k \times m$ Gaussian random matrix $\boldsymbol{\Omega}$ and set

$$\mathbf{F} = \boldsymbol{\Omega}\mathbf{A}.$$

When **A** has exact rank, $\mathbf{A} = \mathbf{EF}$ holds with probability one (for some **E**).

In the general case, the accuracy is as specified by the analysis of the RSVD.

Important: *We do not need to know the factor* **E**. *It just never enters the computation.*

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_{s} and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_{\mathrm{s}})\mathbf{Z}$.

(1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;

(2) Form a $(k + p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega A}$;

(3) Do k steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_{\mathrm{s}})\mathbf{Z}$.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_{s} and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_{\mathrm{s}})\mathbf{Z}$.

(1) Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;

(2) Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega A}$;

(3) Do k steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_{\mathrm{s}})\mathbf{Z}$.

- High flexibility in terms of choice of random matrix. Gaussian works great and is well supported by theory. Sparse random is super fast, and often works well.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

(1) Draw a $(k + p) \times m$ random matrix $\mathbf{\Omega}$;

(2) Form a $(k + p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega A}$;

(3) Do k steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

- High flexibility in terms of choice of random matrix. Gaussian works great and is well supported by theory. Sparse random is super fast, and often works well.
- When a Gaussian distribution is used, power iteration can improve the alignment of $\mathbf{F}$ with the space spanned by the top k right singular vectors.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.

(1) Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;

(2) Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega A}$;

(3) Do k steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

- High flexibility in terms of choice of random matrix. Gaussian works great and is well supported by theory. Sparse random is super fast, and often works well.
- When a Gaussian distribution is used, power iteration can improve the alignment of $\mathbf{F}$ with the space spanned by the top k right singular vectors.
- Step (3) can be modified to use methods other than G-S on the columns:
  - Sophisticated methods like the "strong rank-revealing QR" of Gu and Eisenstat can be used to pick columns that are provably "close" to optimal.
  - The discrete empirical interpolation method (DEIM) of Sorensen and Embree slightly improves on the optimality of the selected indices, at the cost that the full RSVD has to be run first.
  - Remarkably, it is ok to use partially pivoted LU on $\mathbf{F}$. This is *much* faster when the rank is large. "Poor main's DEIM."

Numerical experiments

Question: How should I postprocess the matrix, once I have extracted a sketch?

We will compare:

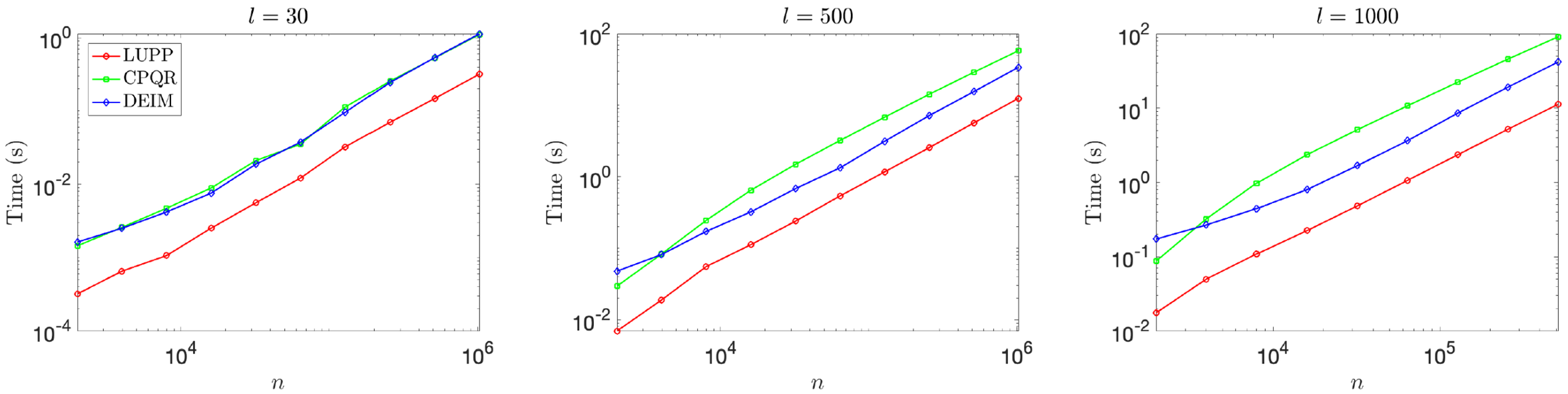
- Column pivoted QR
- DEIM: Form approximate SVD, then partially pivoted LU on the singular vectors.
- Partially pivoted LU directly on the sketching matrix. ("Poor man's DEIM")
- (Form approximate RSVD, then compute "leverage scores", then draw columns based on the leverage scores.)

In these experiments, we use *Gaussian* random matrices.

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *runtime*.



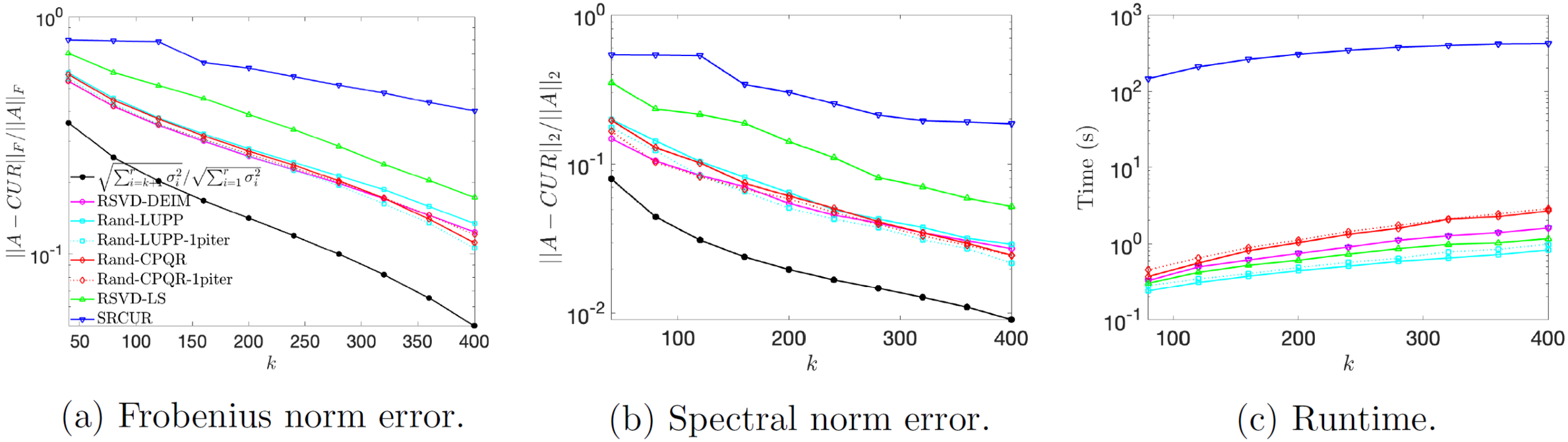
The runtime of various pivoting schemes on the sketches of size $n \times \ell$, scaled with respect to the problem size n, at different embedding dimension ℓ.

Observe that the dimension of the sketch is quite high in the last two examples.

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.



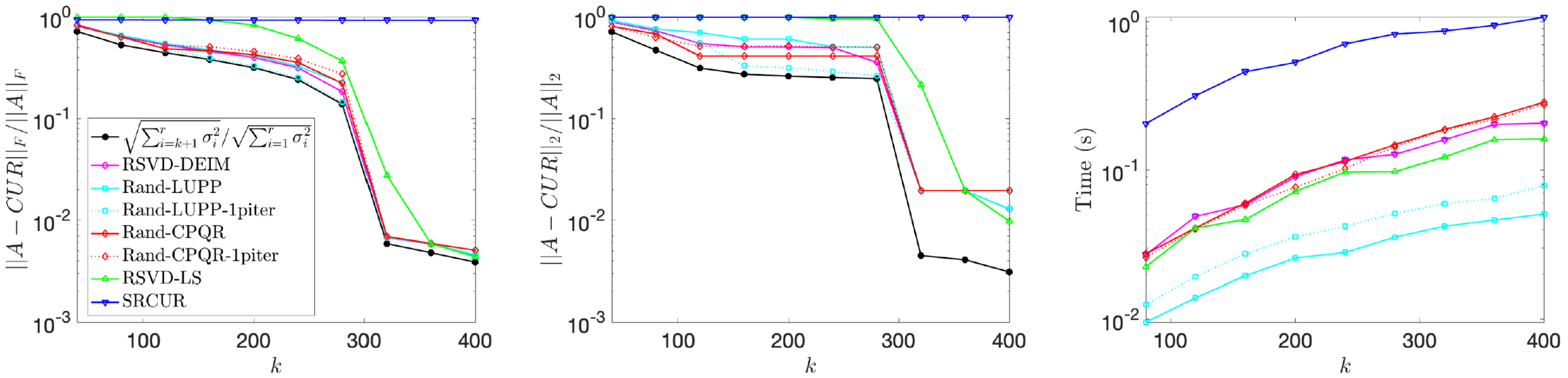
(a) Frobenius norm error. (b) Spectral norm error. (c) Runtime.

*The "*MNIST*" test matrix is dense and of size* $784 \times 60\,000$ *where each column holds one hard drawn digit between 0 and 9. The matrix is* 80% *sparse.*

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.



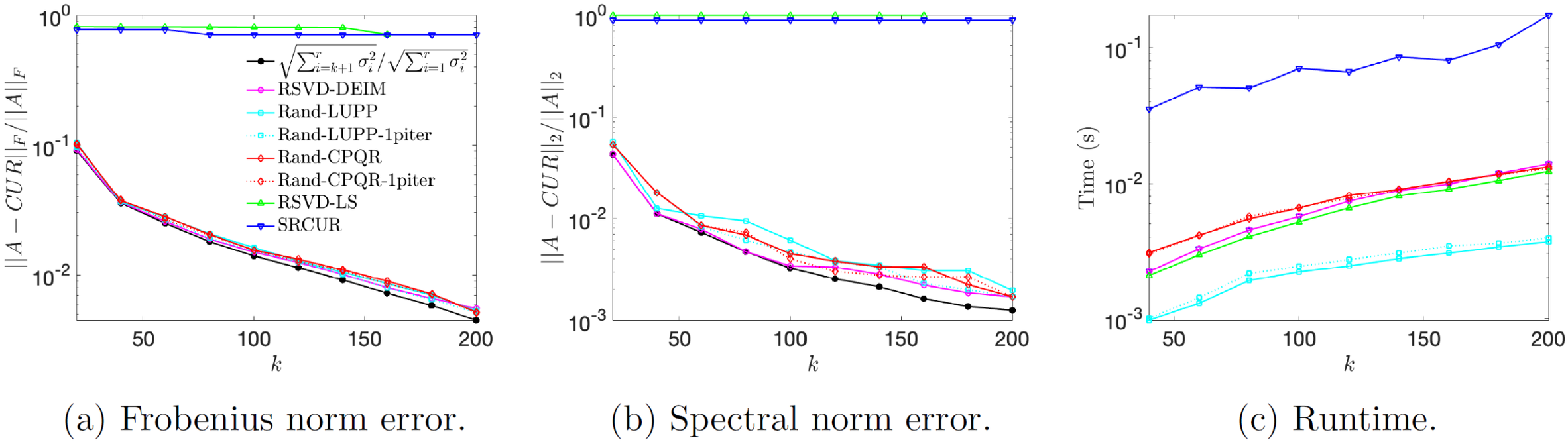
(a) Frobenius norm error. (b) Spectral norm error. (c) Runtime.

*The "*YaleFace64x64*" test matrix holds 165 face images, each with* 64×64 *pixels. The pictures have been normalized, to create a dense matrix of size* 165×4096.

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.



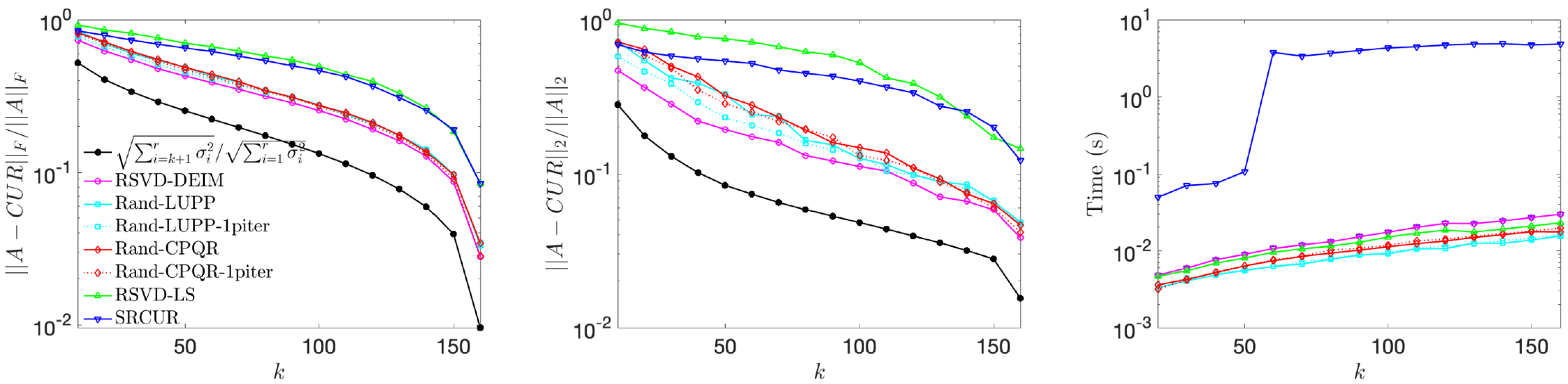
(a) Frobenius norm error. (b) Spectral norm error. (c) Runtime.

*The "*SNN*" test matrix has been used in the CUR literature before. It is an artificial sparse matrix of size* $1\,000 \times 1\,000$.

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.

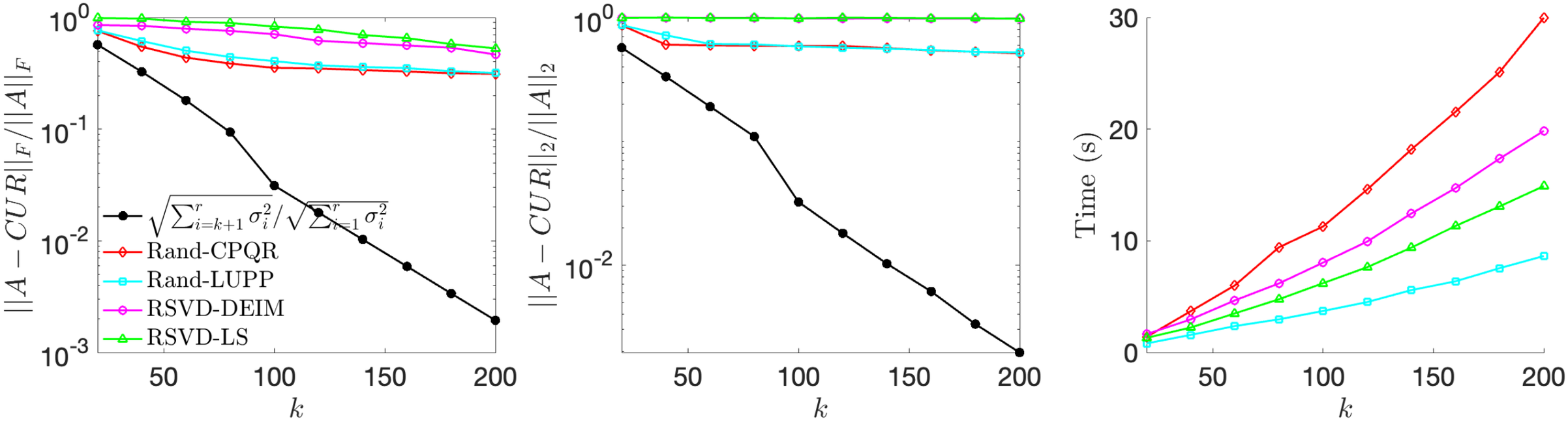


*The "*LARGE*" test matrix is taken from a linear programming example. It is sparse, of size* $4\,282 \times 8\,617$*, with* $20\,635$ *nonzero entries.*

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.



*The "*LARGESPARSE*" test matrix is a synthetic non-negative matrix. It is sparse, of size* $10^6 \times 10^6$.

Take-aways from numerical experiments:

We tested three methods for picking columns from the sketch matrix:

1. Column pivoted QR.
2. DEIM. (Compute approximate RSVD, then do LU with partial pivoting.)
3. Partially pivoted LU directly on the sketch. *It works because of randomization!!*

Observations:

- The three methods are about equally good at picking columns.
  DEIM perhaps slight winner.
- All work quite well in practice.
- In terms of *speed*, partially pivoted LU ("Poor man's DEIM") is the fastest by a margin.
- For very large matrices, you sometimes lose lots of approximation accuracy by using "natural bases" instead of the singular vectors.

Dong & Martinsson, arXiv:2104.05877 [math.NA]

Randomized pivoting in Householder QR

The column selection strategies described can also be applied to resolve a classical problem in numerical linear algebra: How do you pick *groups* of pivots when executing column pivoted QR? The purpose is to move flops from BLAS2 to BLAS3 operations.

Randomized pivoting in Householder QR

Given a dense $n \times n$ matrix $\mathbf{A}$, compute a column pivoted QR factorization

$$\underset{n \times n}{\mathbf{A}} \quad \underset{n \times n}{\mathbf{P}} \approx \underset{n \times n}{\mathbf{Q}} \quad \underset{n \times n}{\mathbf{R}},$$

where, as usual, $\mathbf{Q}$ should be ON, $\mathbf{P}$ is a permutation, and $\mathbf{R}$ is upper triangular.

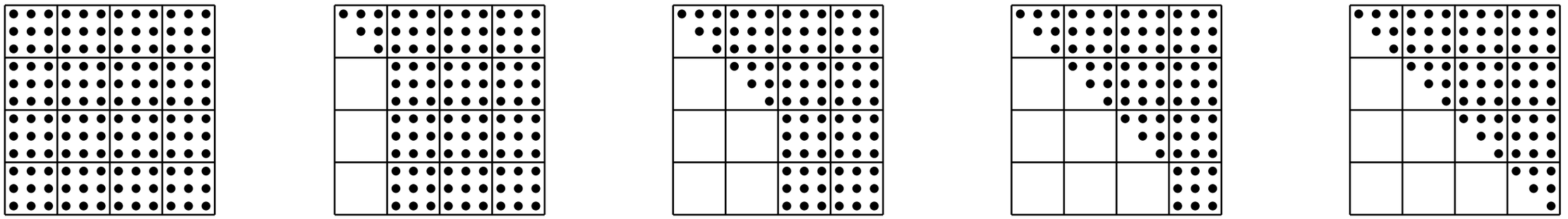
Randomized pivoting in Householder QR

Given a dense $n \times n$ matrix $\mathbf{A}$, compute a column pivoted QR factorization

$$\underset{n \times n}{\mathbf{A}} \quad \underset{n \times n}{\mathbf{P}} \approx \underset{n \times n}{\mathbf{Q}} \quad \underset{n \times n}{\mathbf{R}},$$

where, as usual, $\mathbf{Q}$ should be ON, $\mathbf{P}$ is a permutation, and $\mathbf{R}$ is upper triangular.

The technique proposed is based on a *blocked* version of classical Householder QR:



$$\mathbf{A}_0 = \mathbf{A} \qquad \mathbf{A}_1 = \mathbf{Q}_1^* \mathbf{A}_0 \mathbf{P}_1 \qquad \mathbf{A}_2 = \mathbf{Q}_2^* \mathbf{A}_1 \mathbf{P}_2 \qquad \mathbf{A}_3 = \mathbf{Q}_3^* \mathbf{A}_2 \mathbf{P}_3 \qquad \mathbf{A}_4 = \mathbf{Q}_4^* \mathbf{A}_3 \mathbf{P}_4$$

Each $\mathbf{P}_j$ is a permutation matrix computed via randomized sampling.

Each $\mathbf{Q}_j$ is a product of Householder reflectors.

The key challenge has been to find good permutation matrices.

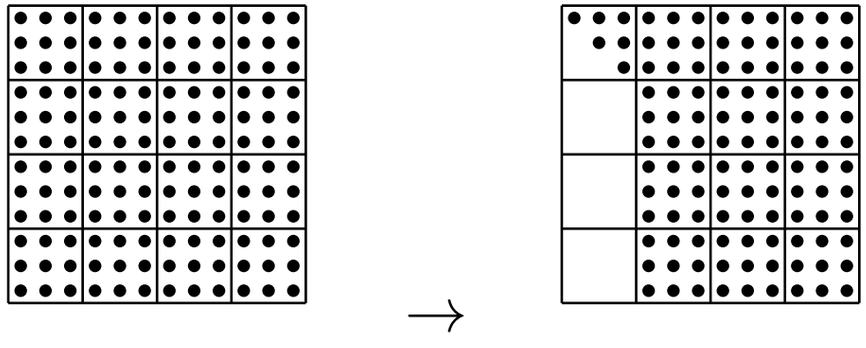
We seek $\mathbf{P}_j$ so that the set of b chosen columns *has maximal spanning volume.*

The pivot selection problem is *very* closely related to the problem of finding spanning columns that we started with! The likelihood that any block of columns is "hit" by the random vectors is directly proportional to its volume. Perfect optimality is *not* required.

Randomized pivoting in Householder QR

How to do block pivoting using randomization:

Let **A** be of size $m \times n$, and let b be a block size.



A $\qquad\qquad$ $\mathbf{Q}^*\mathbf{AP}$

Q is a product of b Householder reflectors.

P is a permutation matrix that moves b "pivot" columns to the leftmost slots.

We seek **P** so that the set of chosen columns *has maximal spanning volume.*

Draw a Gaussian random matrix **G** of size $b \times m$ and form

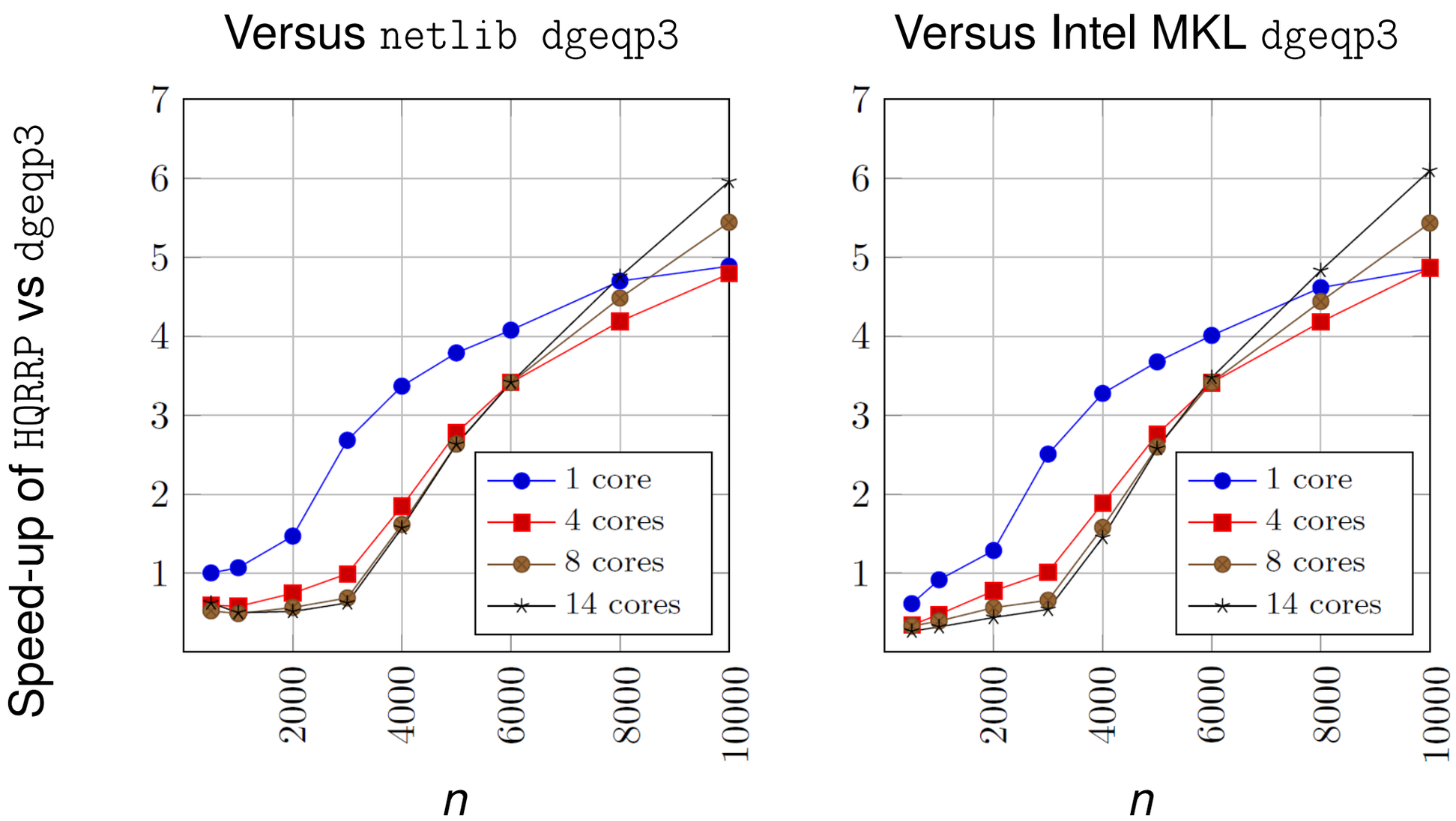
$$\underset{b \times n}{\mathbf{F}} = \underset{b \times m}{\mathbf{G}} \; \underset{m \times n}{\mathbf{A}}$$

The rows of **F** are random linear combinations of the rows of **A**.

Then compute the pivot matrix **P** for the first block by executing traditional column pivoting on the small matrix **F**:

$$\underset{b \times n}{\mathbf{F}} \; \underset{n \times n}{\mathbf{P}} = \underset{b \times b}{\mathbf{Q}_{\text{trash}}} \; \underset{b \times n}{\mathbf{R}_{\text{trash}}}$$

Randomized pivoting in Householder QR



Speedup attained by our randomized algorithm HQRRP for computing a full column pivoted QR factorization of an $n \times n$ matrix. The speed-up is measured versus LAPACK's faster routine dgeqp3 as implemented in Netlib (left) and Intel's MKL (right). Our implementation was done in C, and was executed on an Intel Xeon E5-2695. Joint work with G. Quintana-Ortí, N. Heavner, and R. van de Geijn. Available at: https://github.com/flame/hqrrp/

References: Martinsson arXiv:1505.08115; Duersch/Gu arXiv:1509.06820; Martinsson/Quintana-Ortí/Heavner/van de Geijn SISC 2017; Duersch/Gu SISC 2017 and SIREV 2020.

Epilogue

Nature of different randomized algorithms

Monte Carlo algorithms: Output is a random variable.

- Large variability in output, sensitive to poor random number generators, etc.
- Slow convergence. The error ε scales as $1/\sqrt{n}$ where n is number of instantiations. In consequence, $n \sim 1/\varepsilon^2$ number of samples required.
- *Enable the solution of many otherwise intractable problems.*

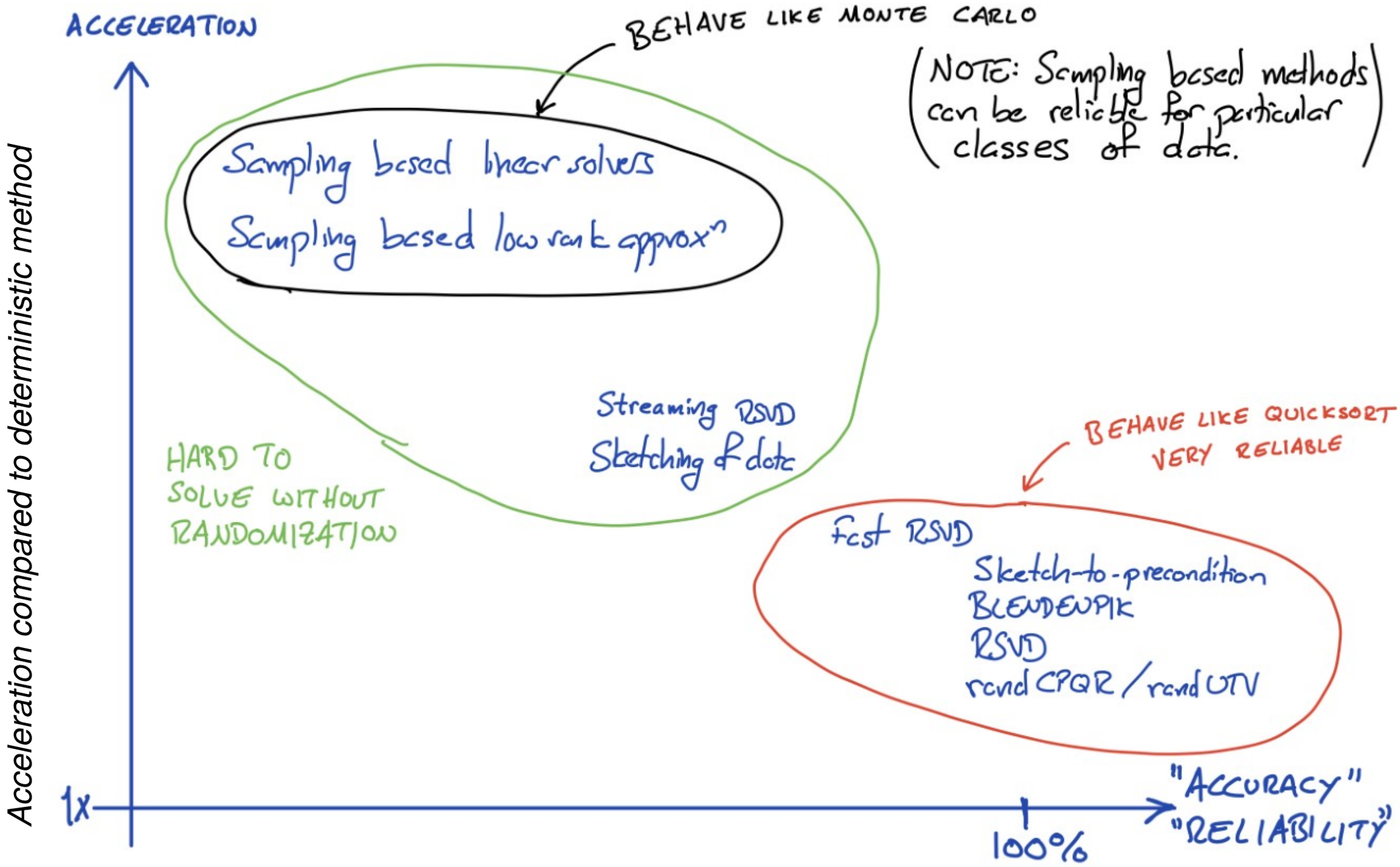
Las Vegas algorithms: Certain to find correct answer. Runtime is stochastic.

- Archetypical example is QuickSort.
- Randomized pre-conditioning is an LV algorithm in that the residual is controlled.

Many of the recently proposed randomized methods are "intermediate".

- Output is a random variable, but errors are very small and concentrated.
- Robust to quality of random number generators, etc.
- Large errors can be caught $\rightarrow$ only problem is occasionally excessive runtime.

Randomized algorithms come with different degrees of acceleration and reliability



Accuracy or reliability compared to deterministic method

This graphic is not to be taken *too* seriously…

Surveys:

- J. Tropp 2019, “Matrix concentration and computational linear algebra.” CMS Lecture Notes 2019-01,
- P. Drineas and M.W. Mahoney, “Lectures on Randomized Numerical Linear Algebra”, Amer. Math. Soc., 2018.
- M.W. Mahoney and P. Drineas, “RandNLA : Randomized Numerical Linear Algebra”, Communications of the ACM, 2016.
- R. Kannan and S. Vempala, “Randomized Algorithms in Numerical Linear Algebra”, Acta Numerica, 2017.
- J. Tropp, “An introduction to matrix concentration inequalities”, Found. Trends Mach. Learn., 2015.
- D. Woodruff, Sketching as a Tool for Numerical Linear Algebra , Foundations and Trends in Theoretical Computer Science, 2014.
- M.W. Mahoney, Randomized Algorithms for Matrices and Data , Foundations and Trends in Machine Learning, 2011.

Surveys directly related to the material presented:

- P.G. Martinsson and J. Tropp, "Randomized Numerical Linear Algebra: Foundations & Algorithms". *Acta Numerica*, 2020. (Arxiv report 2002.01387)

  Long survey summarizing major findings in the field in the past decade.

- P.G. Martinsson, "Randomized methods for matrix computations." The Mathematics of Data, IAS/Park City Mathematics Series, 25(4), pp. 187 - 231, 2018.

  Book chapter that is written to be accessible to a broad audience. Focused on practical aspects.

- 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.

  Survey that describes the randomized SVD and its variations.

Tutorials, summer schools, etc:

- 2020: 3 lecture mini course on randomized linear algebra, KTH, Stockholm. Videos available.
- 2016: Park City Math Institute (IAS): *The Mathematics of Data.*
- 2014: CBMS summer school at Dartmouth College. 10 lectures on YouTube.
- 2009: NIPS tutorial lecture, Vancouver, 2009. Online video available.

Software:

- ID: `http://tygert.com/software.html` (ID, SRFT, CPQR, etc)
- RSVDPACK: `https://github.com/sergeyvoronin` (RSVD, randomized ID and CUR)
- HQRRP: `https://github.com/flame/hqrrp/` (LAPACK compatible randomized CPQR)
- Randomized UTV: `https://github.com/flame/randutv`

DOE report on randomized algorithms: `https://arxiv.org/abs/2104.11079` (2021)