Randomized algorithms for computing full and partial factorizations of matrices

Per-Gunnar Martinsson

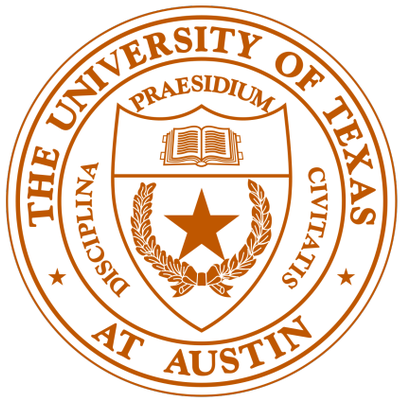
Dept. of Mathematics & Oden Institute for Computational Sciences and Engineering

University of Texas at Austin

Students, postdocs, collaborators: Ke Chen, Yijun Dong, Robert van de Geijn, Abinand Gopal, Nathan Halko, Nathan Heavner, Francisco Igual, James Levitt, Gregorio Quintana-Ortí, Joel Tropp, Sergey Voronin, Bowei Wu, Anna Yesypenko.

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

Research support by:



Outline of talk

1. Introduction to randomized low rank approximation.

2. Interpolatory and CUR factorizations (very brief).

3. Rank revealing factorizations for matrices of full or nearly full rank.

4. Brief survey of related research areas (if time permits):

   ◇ Structured random matrices.

   ◇ Single-view ("streaming") algorithms.

   ◇ Randomized block Krylov methods.

   ◇ Approximation of kernel matrices — $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

   ◇ (Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$.)

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

Randomized SVD (RSVD):

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 $k \ll \min(m, n)$.)

Randomized SVD (RSVD):

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:*

Use randomized projection methods to form an approximate basis for the range of the matrix.

(B) *Deterministic post-processing:*

Restrict the matrix to the subspace determined in Stage A, and perform expensive but accurate computations on the resulting smaller matrix.

Randomized SVD (RSVD):

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

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{U}} \; \underset{k \times k}{\mathbf{D}} \; \underset{k \times n}{\mathbf{V}^*}$$

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 SVD of the matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, D, 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 (RSVD):

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

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{U}} \; \underset{k \times k}{\mathbf{D}} \; \underset{k \times n}{\mathbf{V}^*}$$

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 SVD of the matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, D, 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 (RSVD):

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

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{U}} \; \underset{k \times k}{\mathbf{D}} \; \underset{k \times n}{\mathbf{V}^*}$$

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

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

How does it work? To develop intuition, it helps to first consider the case $\text{rank}(\mathbf{A}) = k$. Then $\text{ran}(\mathbf{Y}) = \text{ran}(\mathbf{A})$ holds with probability 1, so the output is *exactly the SVD* of $\mathbf{A}$. In the general case, contributions from the singular modes beyond the first k will shift $\text{ran}(\mathbf{Y})$ away from the desired space spanned by the dominant k left singular vectors.

Randomized SVD (RSVD):

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

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

Distortions in the randomized projections are fine, since all we need is a subspace that captures "the essential" part of the range. Pollution from unwanted singular modes is harmless, as long as we capture the dominant ones. By drawing p extra samples (for, say, $p = 5$ or $p = 10$), we make the risk of missing anything important essentially zero.

Randomized SVD (RSVD):

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 (RSVD):

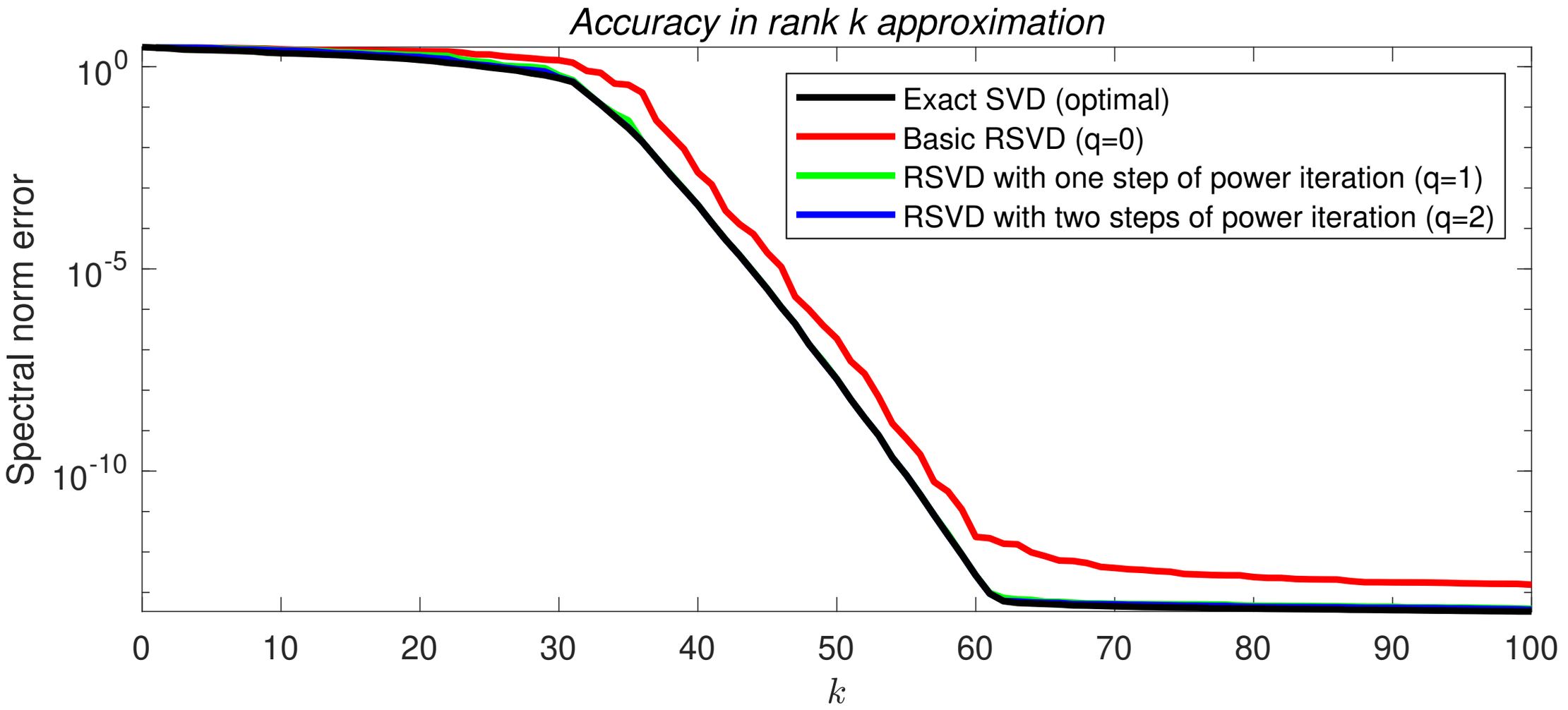
Input: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix $\mathbf{\Omega}$.	(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 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

- For a dense matrix, asymptotic cost is $O(mnk)$, just like column pivoted QR, or a Krylov method. RSVD is faster in practice, since the matrix-matrix multiplication is *very* fast. Acceleration to $O(mn \log(k))$ is possible. (Will discuss later.)

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

- 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 (RSVD):



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

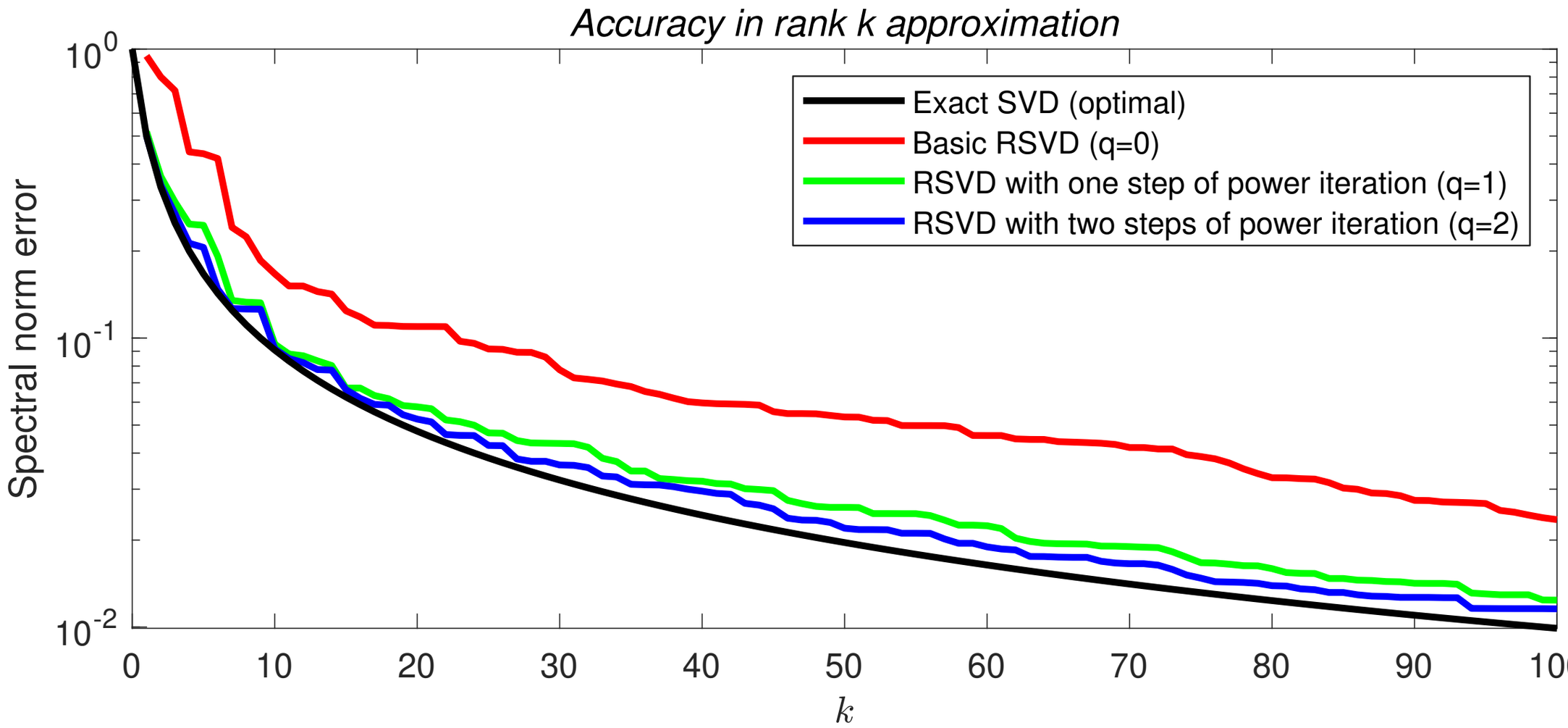
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

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. (Recall that $\mathbf{P}_k\mathbf{A} = \mathbf{U}_k\mathbf{D}_k\mathbf{V}_k^*$.)
The matrix $\mathbf{A}$ is an approximation to a scattering operator for a Helmholtz problem.

Randomized SVD (RSVD):



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

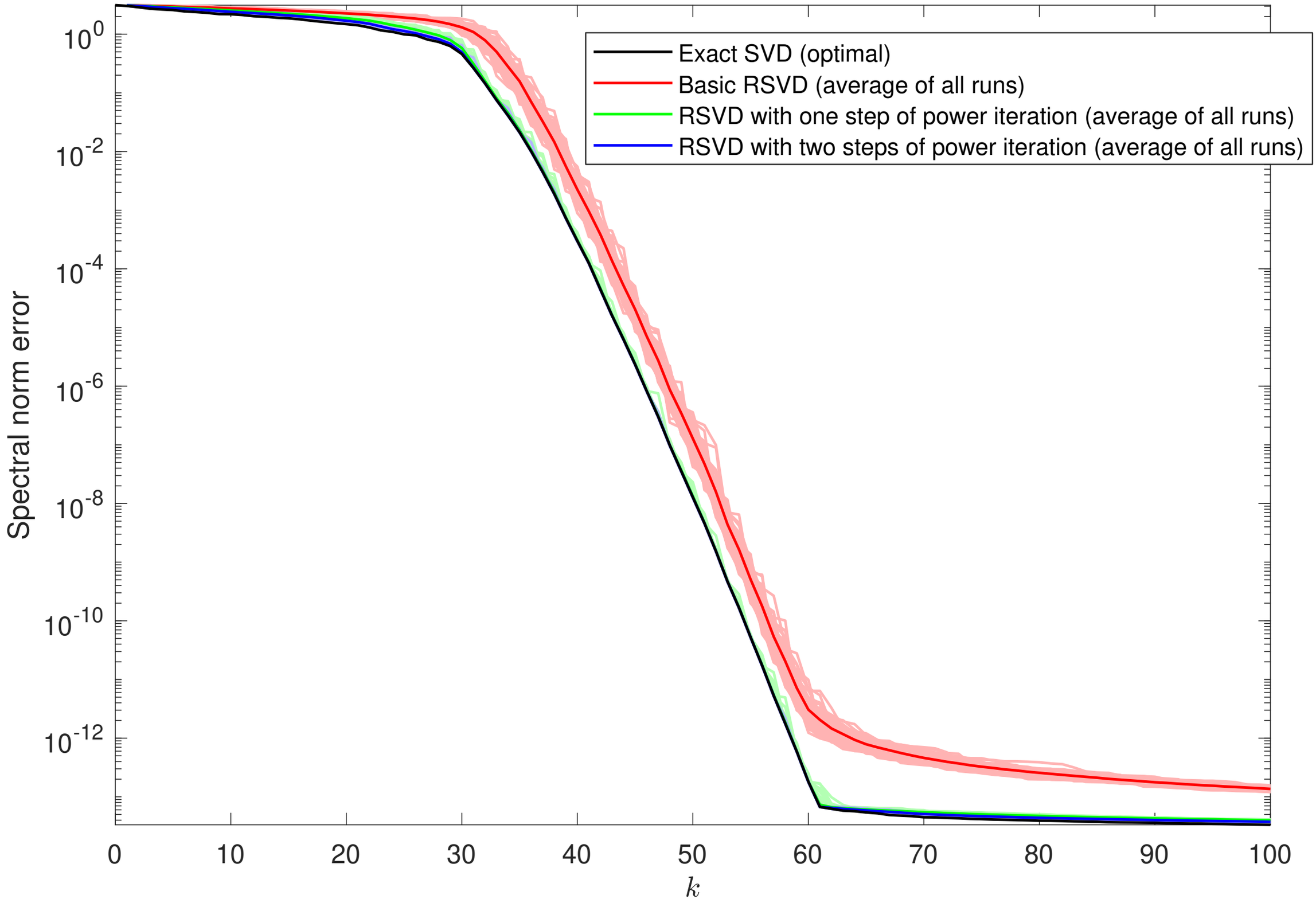
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

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

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

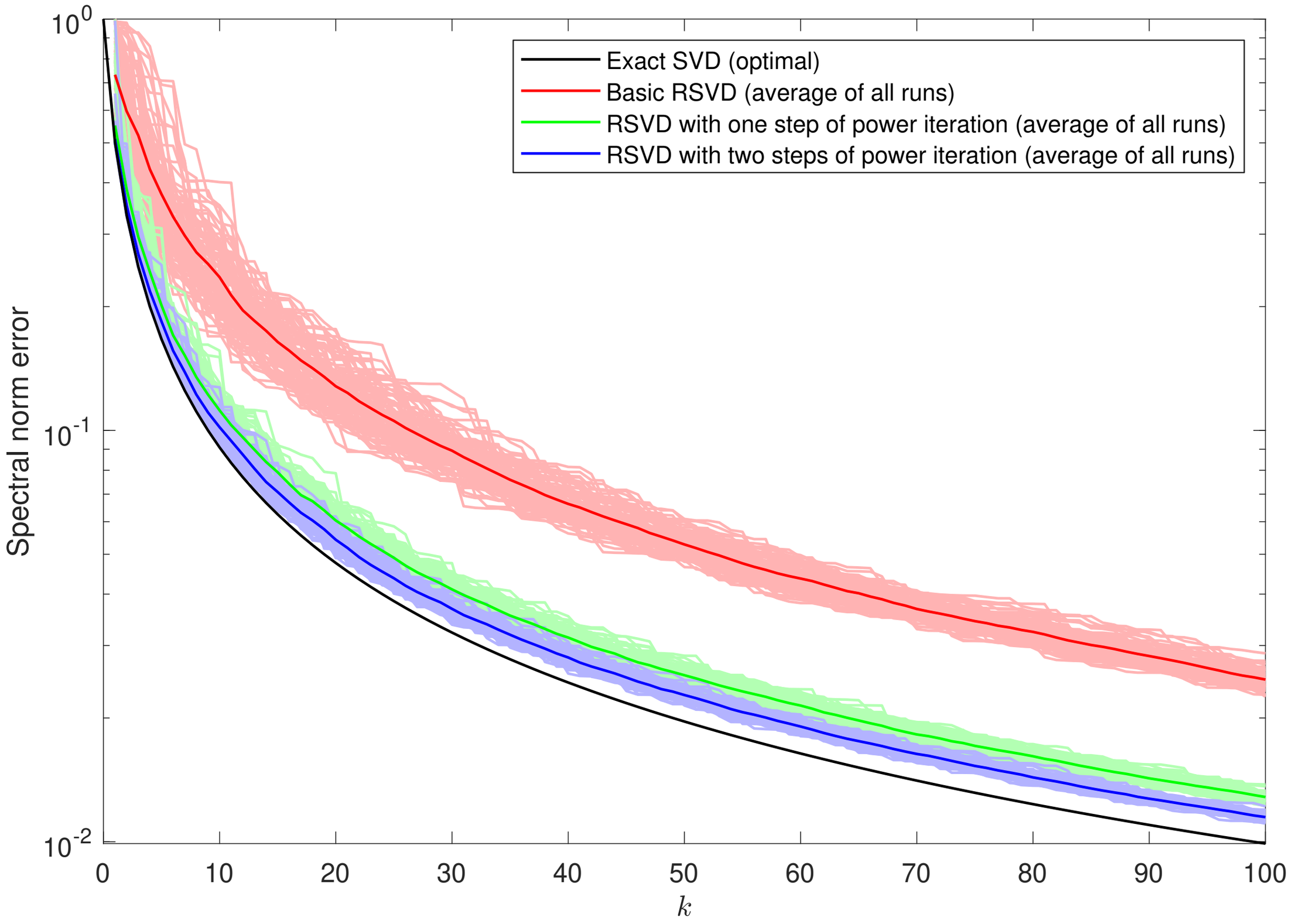
and where $\mathbf{\Omega}$ is a Gaussian random matrix. (Recall that $\mathbf{P}_k\mathbf{A} = \mathbf{U}_k\mathbf{D}_k\mathbf{V}_k^*$.)
The matrix $\mathbf{A}$ now has singular values that decay slowly.

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:

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 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

The output of RSVD is a random variable, as it depends on the draw of $\boldsymbol{\Omega}$. We have rigorous mathematical results describing the errors of the algorithm in expectation, as well as the risk of large deviations. Draw on random matrix theory.

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 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

The output of RSVD is a random variable, as it depends on the draw of $\mathbf{\Omega}$. We have rigorous mathematical results describing the errors of the algorithm in expectation, as well as the risk of large deviations. Draw on random matrix theory.

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 $\mathbf{Q} = \mathtt{orth}(\mathbf{A\Omega})$. 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:

Original work:

- P.G. Martinsson, V. Rokhlin and M. Tygert (2006a), A randomized algorithm for the approximation of matrices, Technical Report Yale CS research report YALEU/DCS/RR-1361, Yale.
- Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert, *Randomized algorithms for the low-rank approximation of matrices*. Proceedings of the National Academy of Sciences 2007 104: 20167-20172.
- V. Rokhlin, A. Szlam, and M. Tygert, *A Randomized Algorithm for Principal Component Analysis*, SIAM J. Matrix Anal. Appl., 31(3), 1100–1124.
- P.G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the approximation of matrices*. Applied and Computational Harmonic Analysis, 30(1), pp. 47–68, 2011.
- F. Woolfe, E. Liberty, V. Rokhlin, M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Applied and Computational Harmonic Analysis, **25**(3), 2008.

Relevant prior work:

- C. H. Papadimitriou, P. Raghavan, H. Tamaki and S. Vempala (2000), *Latent semantic indexing: a probabilistic analysis*, Vol. 61, pp. 217–235.
- A. Frieze, R. Kannan and S. Vempala (2004), *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM 51(6), 1025–1041.

Surveys:

- N. Halko, P. G. Martinsson, J. A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, 2011.
- P.G. Martinsson, *Randomized Methods for Matrix Computations*. In the 2018 book *The Mathematics of Data,* published by AMS.

Outline of talk

1. Introduction to randomized low rank approximation. *Done!*
2. Interpolatory and CUR factorizations (very brief).
3. Rank revealing factorizations for matrices of full or nearly full rank.
4. Brief survey of related research areas:
   - ◇ Structured random matrices.
   - ◇ Single-view ("streaming") algorithms.
   - ◇ Randomized block Krylov methods.
   - ◇ Approximation of kernel matrices — $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
   - ◇ (Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$.)

Interpolatory and CUR factorizations

Let $\mathbf{A}$ be an $m \times n$ matrix of approximate rank k. We seek a factorization

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{X}} \quad \underset{k \times n}{\mathbf{R}},$$

where $\mathbf{R}$ holds a subset of k of the rows of $\mathbf{A}$.

In other words, $\mathbf{R} = \mathbf{A}(I_{\mathrm{S}}, :)$ for some index vector I_{S} of length k. We seek $\{I_{\mathrm{S}}, \mathbf{X}\}$.

Why? Data interpretation, computational efficiency, preserve sparsity/non-negativity, ...

Interpolatory and CUR factorizations

Let **A** be an $m \times n$ matrix of approximate rank k. We seek a factorization

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{X}} \underset{k \times n}{\mathbf{R}},$$

where **R** holds a subset of k of the rows of **A**.

In other words, $\mathbf{R} = \mathbf{A}(I_S, :)$ for some index vector I_S of length k. We seek $\{I_S, \mathbf{X}\}$.

Why? Data interpretation, computational efficiency, preserve sparsity/non-negativity, ...

Deterministic techniques

- Finding the absolutely *optimal* set I_S is a very hard problem.
  (Typically: "optimal" = "maximal spanning volume")
- In practice, Gram-Schmidt on the rows works very well (column pivoted QR on $\mathbf{A}^t$).
- Sophisticated pivoting strategies in QR are guaranteed to lead to close to optimal choices. [Gu, Eisenstat 1996]

Interpolatory and CUR factorizations

Let $\mathbf{A}$ be an $m \times n$ matrix of approximate rank k. We seek a factorization

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{X}} \underset{k \times n}{\mathbf{R}},$$

where $\mathbf{R}$ holds a subset of k of the rows of $\mathbf{A}$.

In other words, $\mathbf{R} = \mathbf{A}(I_{\mathrm{S}}, :)$ for some index vector I_{S} of length k. We seek $\{I_{\mathrm{S}}, \mathbf{X}\}$.

Why? Data interpretation, computational efficiency, preserve sparsity/non-negativity, ...

Randomized strategy:

- Draw a tall thin Gaussian matrix $\mathbf{\Omega}$ of size, say $n \times (k + 10)$.
- Form the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.
- Perform Gram-Schmidt on the rows of $\mathbf{Y}$. Compute $\{I_{\mathrm{S}}, \mathbf{X}\}$ from $\mathbf{Y}$!

  If $\mathbf{Y} \approx \mathbf{XY}(I_{\mathrm{S}}, :)$, then "automatically", $\mathbf{A} \approx \mathbf{XA}(I_{\mathrm{S}}, :)$!

Notes:

- Only one interaction with $\mathbf{A}$. (As opposed to two in RSVD.)
- Can be accelerated to complexity $O(mn \log(k))$ by using a "structured" $\mathbf{\Omega}$. (Later!)
- An ID can easily be converted to an SVD. Shortcut for RSVD!

Interpolatory and CUR factorizations, variations: $\mathbf{A}$ is of size $m \times n$ and rank $\approx k$.

Row ID (what we discussed so far):

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{X}} \quad \underset{k \times n}{\mathbf{R}},$$

where $\mathbf{R} = \mathbf{A}(I_{\mathrm{S}}, :)$ for some index vector I_{S} that identifies k rows.

Column ID:

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

where $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$ for some index vector J_{S} that identifies k columns.

Double-sided ID:

$$\underset{m \times n}{\mathbf{A}} \quad \approx \quad \underset{m \times k}{\mathbf{X}} \quad \mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}}) \quad \underset{k \times n}{\mathbf{Z}}.$$

The vectors I_{S} and J_{S}, and the matrices $\mathbf{X}$ and $\mathbf{Z}$ are as above.

CUR factorization:

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

where $\mathbf{C}$ and $\mathbf{R}$ are as above. For instance, $\mathbf{U} = \mathbf{C}^{\dagger}\mathbf{A}\mathbf{R}^{\dagger}$, or $\mathbf{U} = \mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}})^{-1}$.

Any of the factorizations can be computed with the randomized technique described.

Interpolatory and CUR factorizations, strategies based on sampling

Many popular algorithms for computing the CUR factorization are based on *randomized sampling*. (As opposed to the random embeddings discussed so far.)

The idea is to draw subsets of columns and rows, given some probability distribution.

So called *leverage scores* provide sampling probabilities that are in some sense ideal. Powerful theory has been developed.

Computing the leverage scores can be expensive, however. Computationally efficient ways to estimate them have been a subject of much research. Iterative methods can be very helpful.

Sampling is a particularly appealing strategy in cases where the matrix is very large, or expensive to form explicitly. Has been used successfully for, e.g., solving kernel ridge regression problems.

Interpolatory and CUR factorizations

References:

- M. W. Mahoney and P. Drineas *CUR matrix decompositions for improved data analysis*, PNAS, 2009, **106**(3).
- S.A. Goreinov, N.L. Zamarashkin, E.E. Tyrtyshnikov, *Pseudo-skeleton approximations by matrices of maximal volume*, **62**(4), 1997.
- M. Gu, S. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM Journal on Scientific Computing, **17**(4), 1996.
- E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, and M. Tygert, "Randomized algorithms for the low-rank approximation of matrices". *PNAS*, **104**(51), 2007.
- P.G. Martinsson, V. Rokhlin, and M. Tygert, *On interpolation and integration in finite-dimensional spaces of bounded functions*, Communications in Applied Math. and Comp. Science, **1**, 2006.
- N. Halko, P. G. Martinsson, J. A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, 2011.
- S. Wang, Z. Zhang, *Improving CUR matrix decomposition and the Nyström approximation via adaptive sampling*, The Journal of Machine Learning Research, **14**(1), 2013.
- P.G. Martinsson and S. Voronin, *Efficient algorithms for CUR and interpolative matrix decompositions*, Advances in Computational Mathematics, 43(3), pp. 495-516, 2017.
- P.G. Martinsson, *Randomized methods for matrix computations*, The Mathematics of Data, IAS/Park City Mathematics Series, 25(4), pp. 187 - 231, 2018.
- D.J. Biagioni, D. Beylkin, G. Beylkin, *Randomized interpolative decomposition of separated representations* Journal of Computational Physics, **281**(15), 2015.

Outline of talk

1. Introduction to randomized low rank approximation. *Done!*
2. Interpolatory and CUR factorizations (very brief). *Done!*
3. Rank revealing factorizations for matrices of full or nearly full rank.
4. Brief survey of related research areas (if time permits):
   - ◇ Structured random matrices.
   - ◇ Single-view ("streaming") algorithms.
   - ◇ Randomized block Krylov methods.
   - ◇ Approximation of kernel matrices — $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
   - ◇ (Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$.)

Computing full (or nearly full) rank revealing factorizations of matrices

Suppose **A** is a dense $n \times n$ matrix, and that we seek a *full* rank-revealing factorization.

We use the term "rank-revealing" informally, and take it to mean only that a truncated factorization is a reasonably close to optimal low rank approximation to the matrix. In this sense, we can classify some standard factorizations as follows:

Not rank-revealing	Rank-revealing
• Unpivoted QR (QR) • Partially pivoted LU	• Column pivoted QR (CPQR) • Fully pivoted LU • SVD

Computing full (or nearly full) rank revealing factorizations of matrices

Suppose **A** is a dense $n \times n$ matrix, and that we seek a *full* rank-revealing factorization.

We use the term "rank-revealing" informally, and take it to mean only that a truncated factorization is a reasonably close to optimal low rank approximation to the matrix. In this sense, we can classify some standard factorizations as follows:

Not rank-revealing	Rank-revealing
• Unpivoted QR (QR) • Partially pivoted LU	• Column pivoted QR (CPQR) • Fully pivoted LU • SVD
Fast.	Slow.

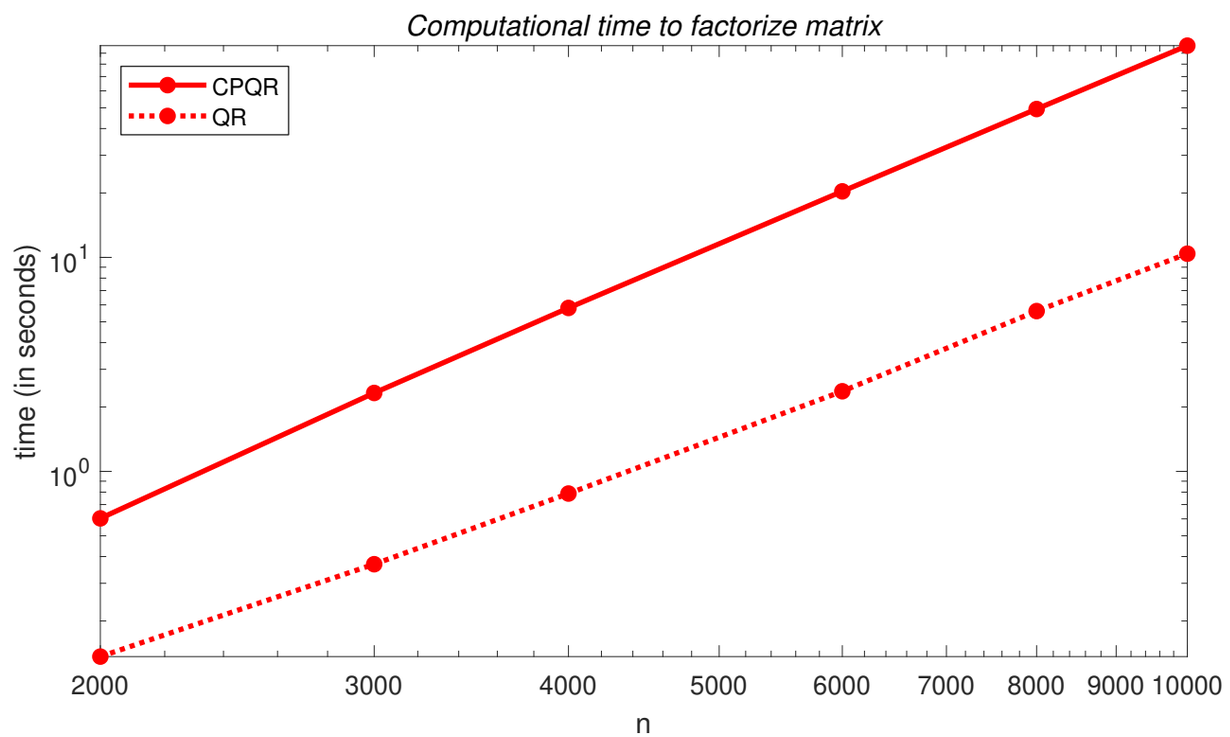
The rank revealing factorizations to the right all depend on algorithms that proceed through a sequence of rank-one updates to the matrix. This makes them slow when executed on modern hardware (even on a single core).

Computing full (or nearly full) rank revealing factorizations of matrices

Suppose **A** is a dense $n \times n$ matrix, and that we seek a *full* rank-revealing factorization.

We use the term "rank-revealing" informally, and take it to mean only that a truncated factorization is a reasonably close to optimal low rank approximation to the matrix. In this sense, we can classify some standard factorizations as follows:

Not rank-revealing	Rank-revealing
• Unpivoted QR (QR) • Partially pivoted LU	• Column pivoted QR (CPQR) • Fully pivoted LU • SVD
Fast.	Slow.



Computing full (or nearly full) rank revealing factorizations of matrices

The reason CPQR is so much slower than QR is that it relies on a sequence of rank-one updates.

Computing full (or nearly full) rank revealing factorizations of matrices

The reason CPQR is so much slower than QR is that it relies on a sequence of rank-one updates.

Randomization to the rescue! D. Stott Parker (1995) proposed an elegant solution:

(1) Randomly mix the columns by right multiplying **A** by a random unitary matrix **V**:

$$\mathbf{A}_{\text{rand}} = \mathbf{AV}.$$

(2) Perform unpivoted QR on the new matrix

$$\mathbf{A}_{\text{rand}} = \mathbf{UR}$$

The resulting factorization

$$\mathbf{A} = \mathbf{A}_{\text{rand}}\mathbf{V}^* = \mathbf{URV}^*$$

is provably "rank-revealing" and leads to stable linear solves.

For computational efficiency, Parker introduced a random structured matrix (a bit ahead of the times) called a "random butterfly transform".

Further refinements — Demmel, Dumitriu, Holtz, Grigori, Dongarra, etc.

Computing full (or nearly full) rank revealing factorizations of matrices

Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw a Gaussian random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{\Omega}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{V}\mathbf{R}_{\text{trash}}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorization

$$\mathbf{A} = \big(\mathbf{AV}\big)\mathbf{V}^* = \mathbf{URV}^*$$

that is excellent at revealing the rank of $\mathbf{A}$. Faster than CPQR, despite far more flops.

Computing full (or nearly full) rank revealing factorizations of matrices

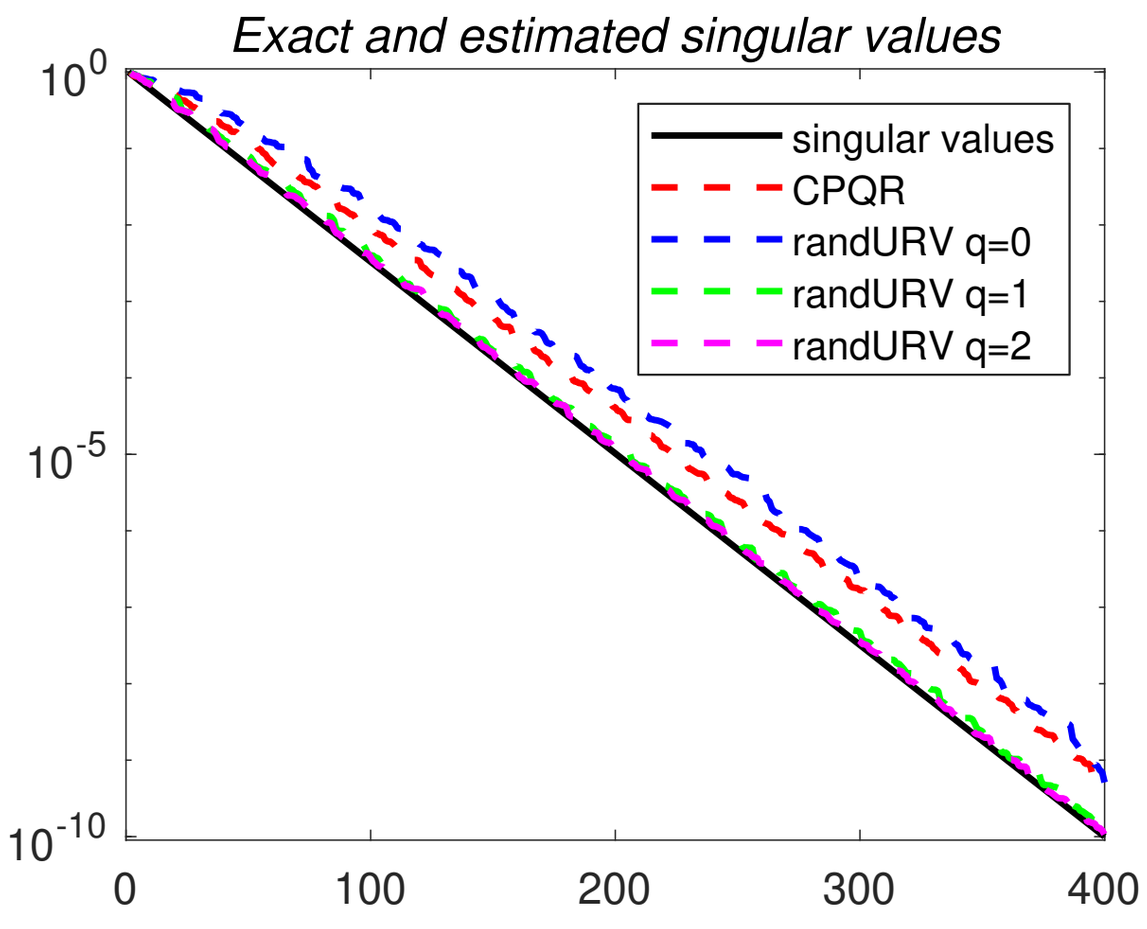
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

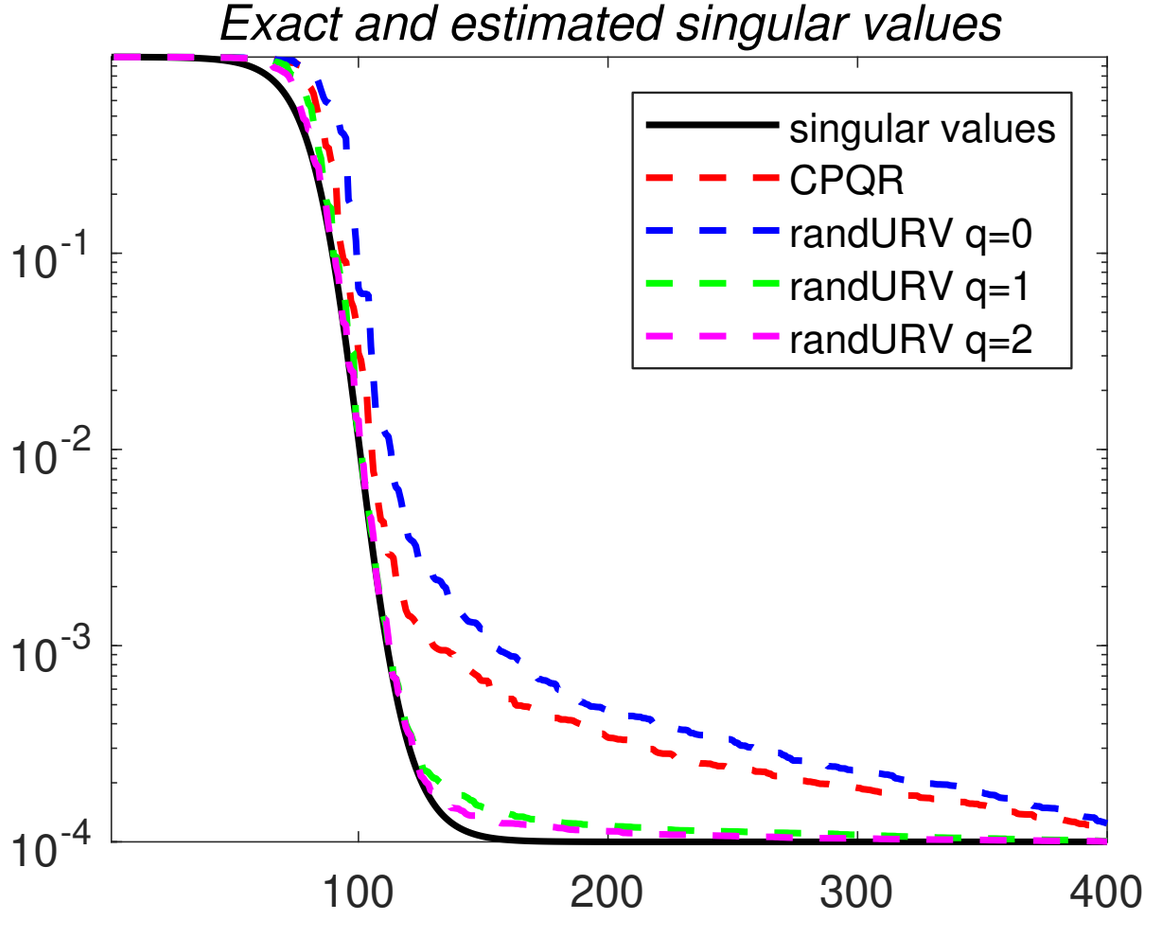
1. Draw a Gaussian random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{\Omega}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{V}\mathbf{R}_{\text{trash}}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorization

$$\mathbf{A} = \big(\mathbf{AV}\big)\mathbf{V}^* = \mathbf{URV}^*$$

that is excellent at revealing the rank of $\mathbf{A}$. Faster than CPQR, despite far more flops.





Computing full (or nearly full) rank revealing factorizations of matrices

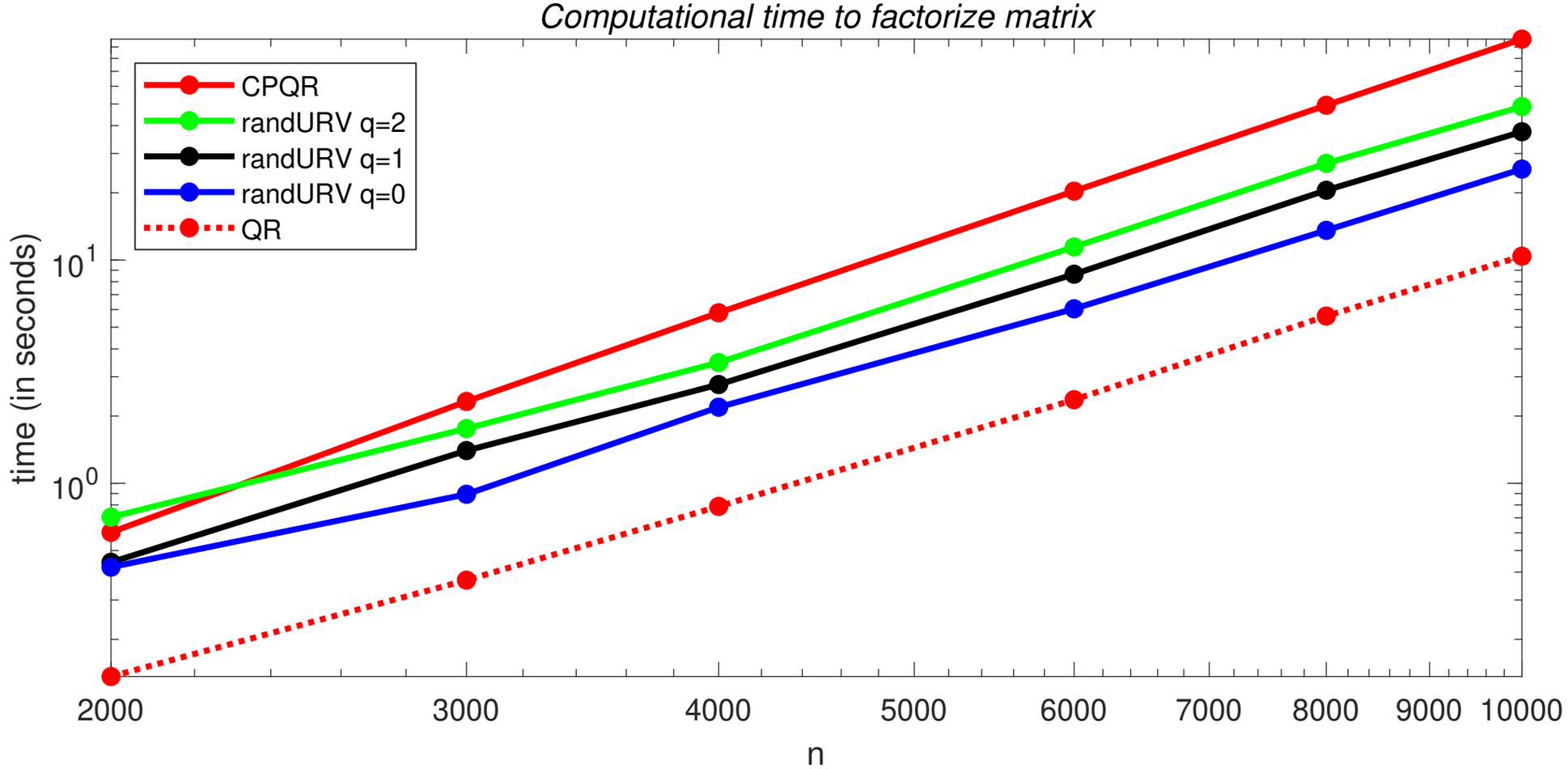
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw a Gaussian random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{\Omega}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{V}\mathbf{R}_{\text{trash}}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorization

$$\mathbf{A} = \big(\mathbf{AV}\big)\mathbf{V}^* = \mathbf{URV}^*$$

that is excellent at revealing the rank of $\mathbf{A}$. Faster than CPQR, despite far more flops.



Computing full (or nearly full) rank revealing factorizations of matrices

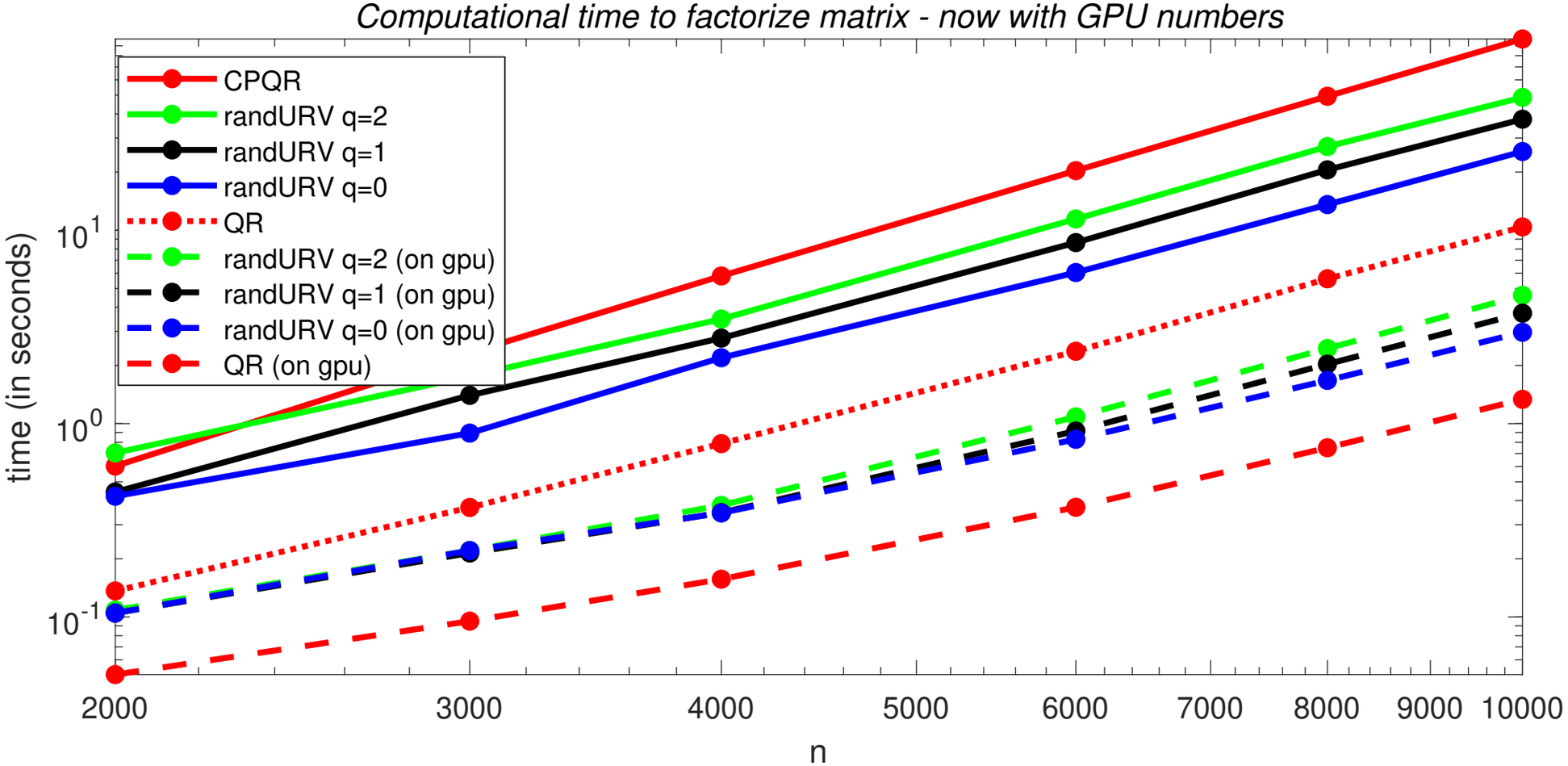
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw a Gaussian random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{\Omega}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{V}\mathbf{R}_{\text{trash}}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorization

$$\mathbf{A} = \big(\mathbf{AV}\big)\mathbf{V}^* = \mathbf{URV}^*$$

that is excellent at revealing the rank of $\mathbf{A}$. Faster than CPQR, despite far more flops.



Computing full (or nearly full) rank revealing factorizations of matrices

Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw a Gaussian random matrix $\mathbf{\Omega}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{\Omega}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{V}\mathbf{R}_{\text{trash}}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorization

$$\mathbf{A} = \big(\mathbf{AV}\big)\mathbf{V}^* = \mathbf{URV}^*$$

that is excellent at revealing the rank of $\mathbf{A}$. Faster than CPQR, despite far more flops.

The method is extremely simple to code:

```
G = randn(n);
for j = 1:q
  G = A'*(A*G);
end
[V,~] = qr(G);
[U,R] = qr(A*V);
```

Computing full (or nearly full) rank revealing factorizations of matrices

Next: A randomized technique for computing a *column pivoted QR* factorization.

- Close to the speed of *unpivoted* QR.
- Pivot selection quality similar to classical CPQR.
- Same asymptotic flop count as classical QR.
- Can be stopped when a given tolerance is met, to produce a partial factorization.

Note: All methods discussed have complexity $O(n^3)$. We are discussing the scaling factor.

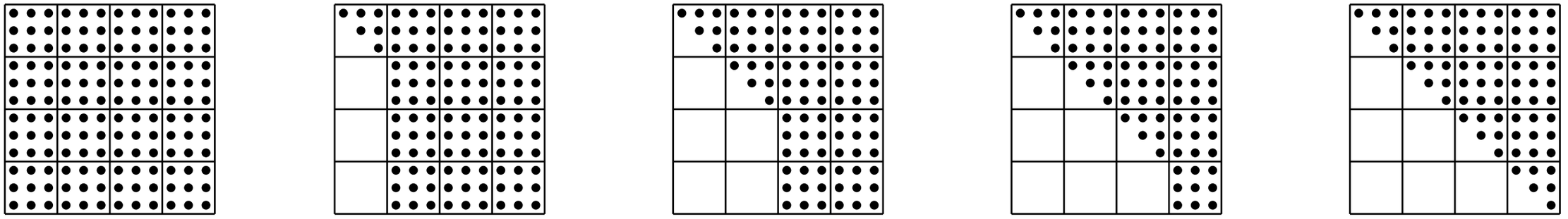
Computing full (or nearly full) rank revealing factorizations of matrices

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.

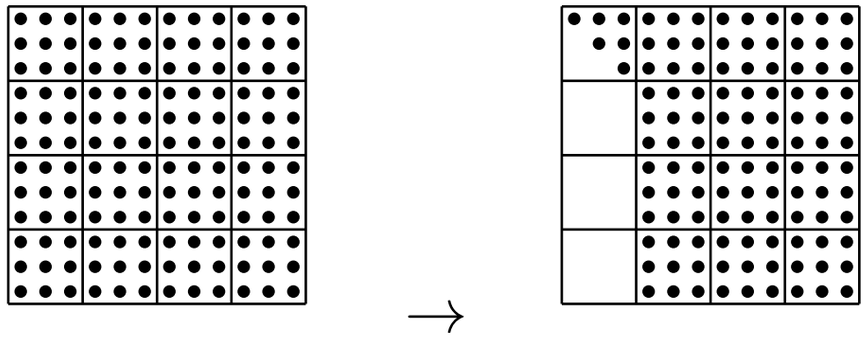
We seek $\mathbf{P}_j$ so that the set of b chosen columns *has maximal spanning volume.*

Perfect for randomized sampling! The likelihood that any block of columns is "hit" by the random vectors is directly proportional to its volume. Perfect optimality is *not* required.

Computing full (or nearly full) rank revealing factorizations of matrices

How to do block pivoting using randomization:

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



A $\quad\rightarrow\quad$ $\mathbf{Q}^*\mathbf{AP}$

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 $\mathbf{\Omega}$ of size $b \times m$ and form

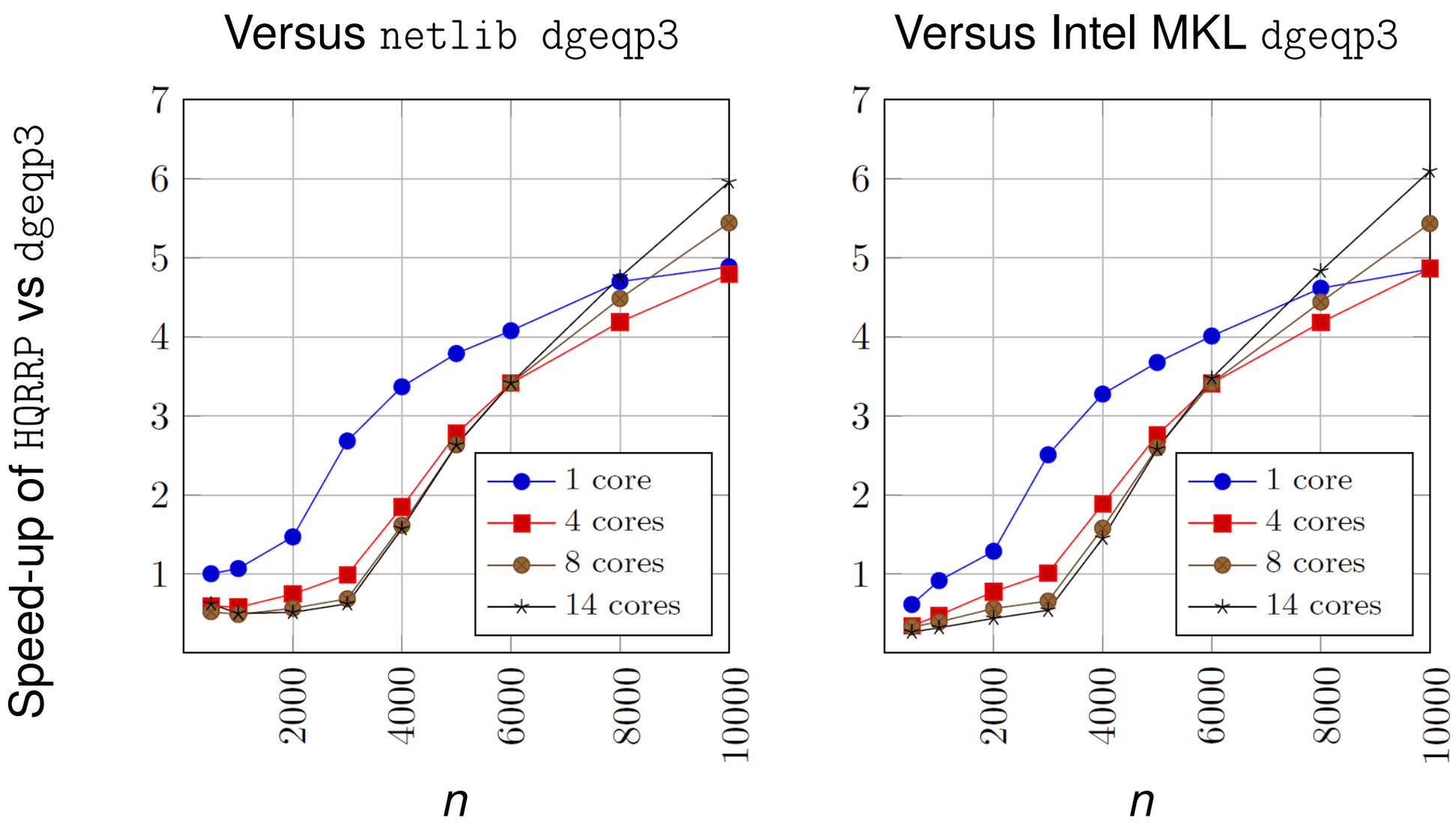
$$\underset{b \times n}{\mathbf{Y}} = \underset{b \times m}{\mathbf{\Omega}} \; \underset{m \times n}{\mathbf{A}}$$

The rows of **Y** 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 **Y**:

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

Computing full (or nearly full) rank revealing factorizations of matrices



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/`

Computing full (or nearly full) rank revealing factorizations of matrices

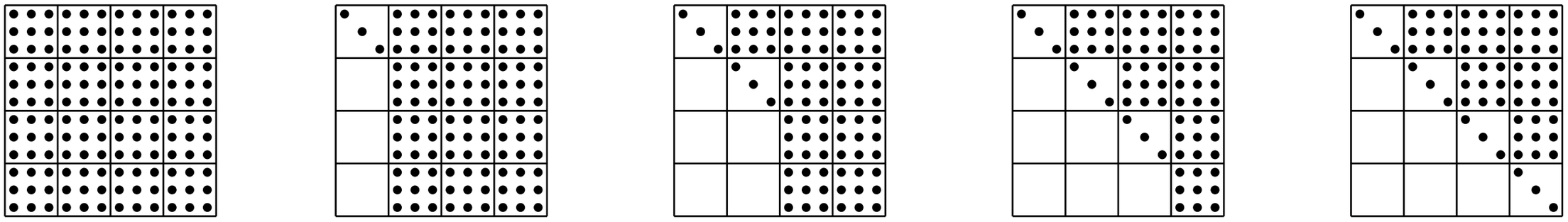
Given a dense $n \times n$ matrix $\mathbf{A}$, compute a factorization

$$\underset{n \times n}{\mathbf{A}} = \underset{n \times n}{\mathbf{U}} \quad \underset{n \times n}{\mathbf{T}} \quad \underset{n \times n}{\mathbf{V}^*},$$

where $\mathbf{T}$ is upper triangular, $\mathbf{U}$ and $\mathbf{V}$ are unitary.

Observe: More general than CPQR since we used to insist that $\mathbf{V}$ be a permutation.

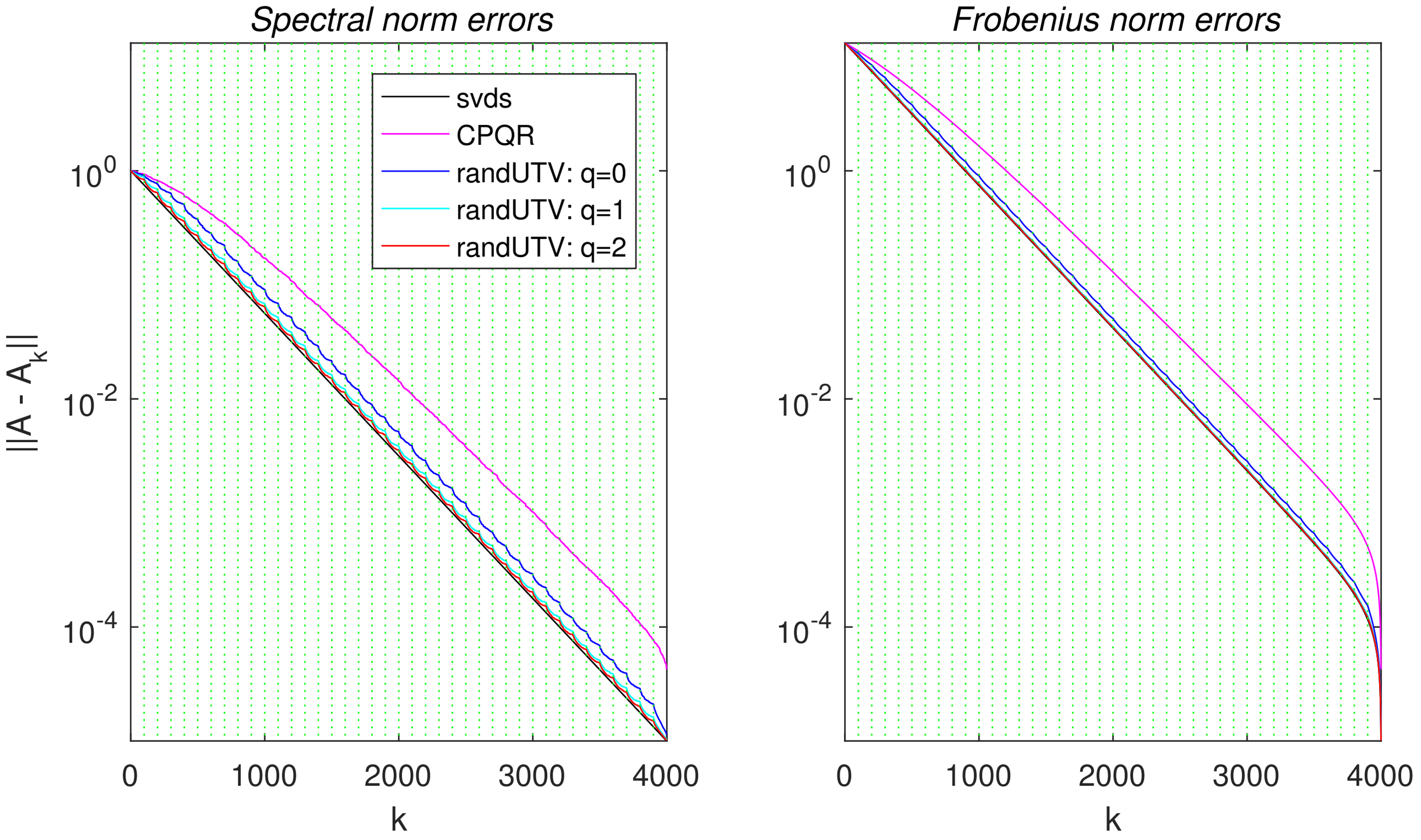
The technique proposed is based on a blocked version of classical Householder QR:



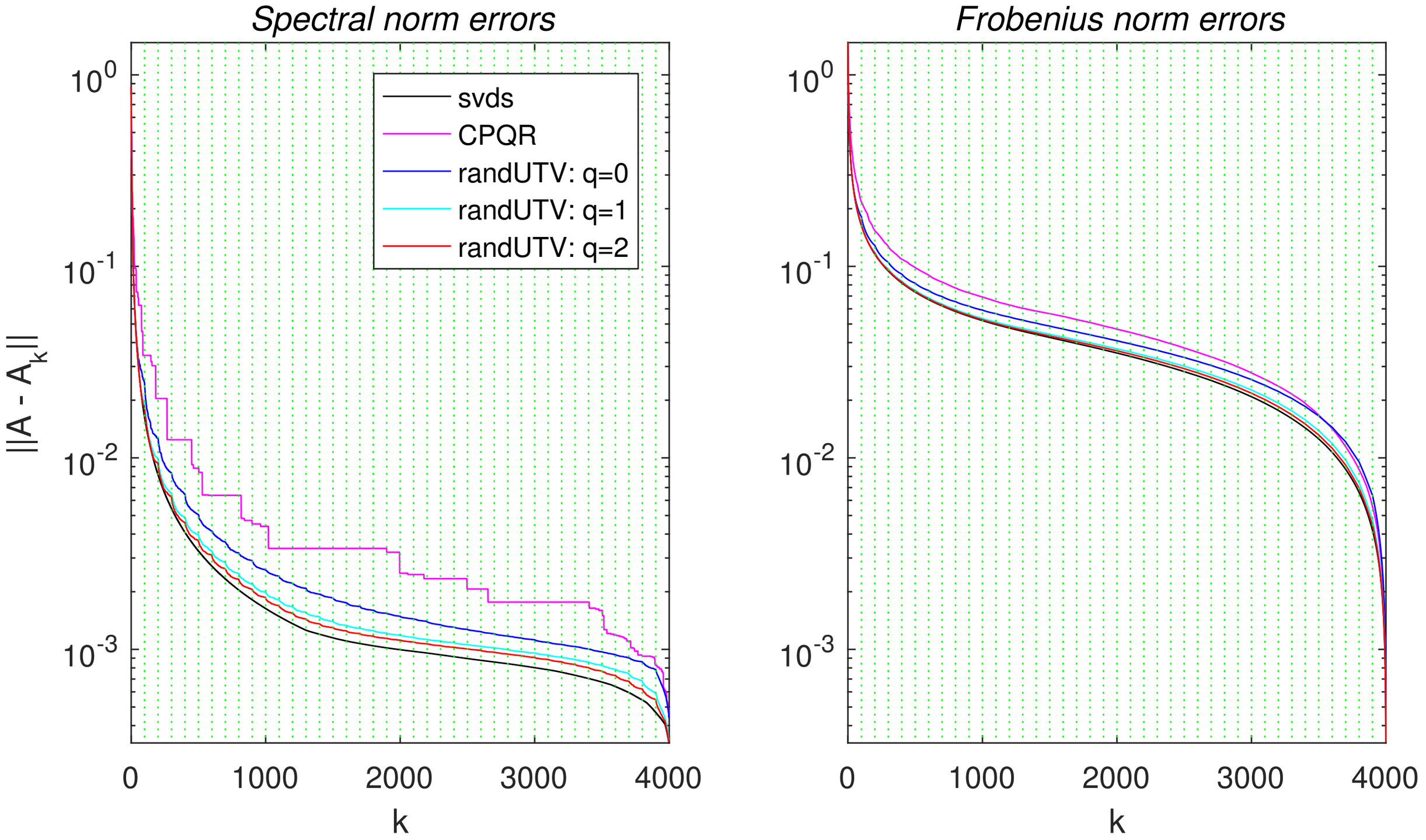
$$\mathbf{A}_0 = \mathbf{A} \qquad \mathbf{A}_1 = \mathbf{U}_1^* \mathbf{A}_0 \mathbf{V}_1 \qquad \mathbf{A}_2 = \mathbf{U}_2^* \mathbf{A}_1 \mathbf{V}_2 \qquad \mathbf{A}_3 = \mathbf{U}_3^* \mathbf{A}_2 \mathbf{V}_3 \qquad \mathbf{A}_4 = \mathbf{U}_4^* \mathbf{A}_3 \mathbf{V}_4$$

Both $\mathbf{U}_j$ and $\mathbf{V}_j$ are (mostly...) products of b Householder reflectors.

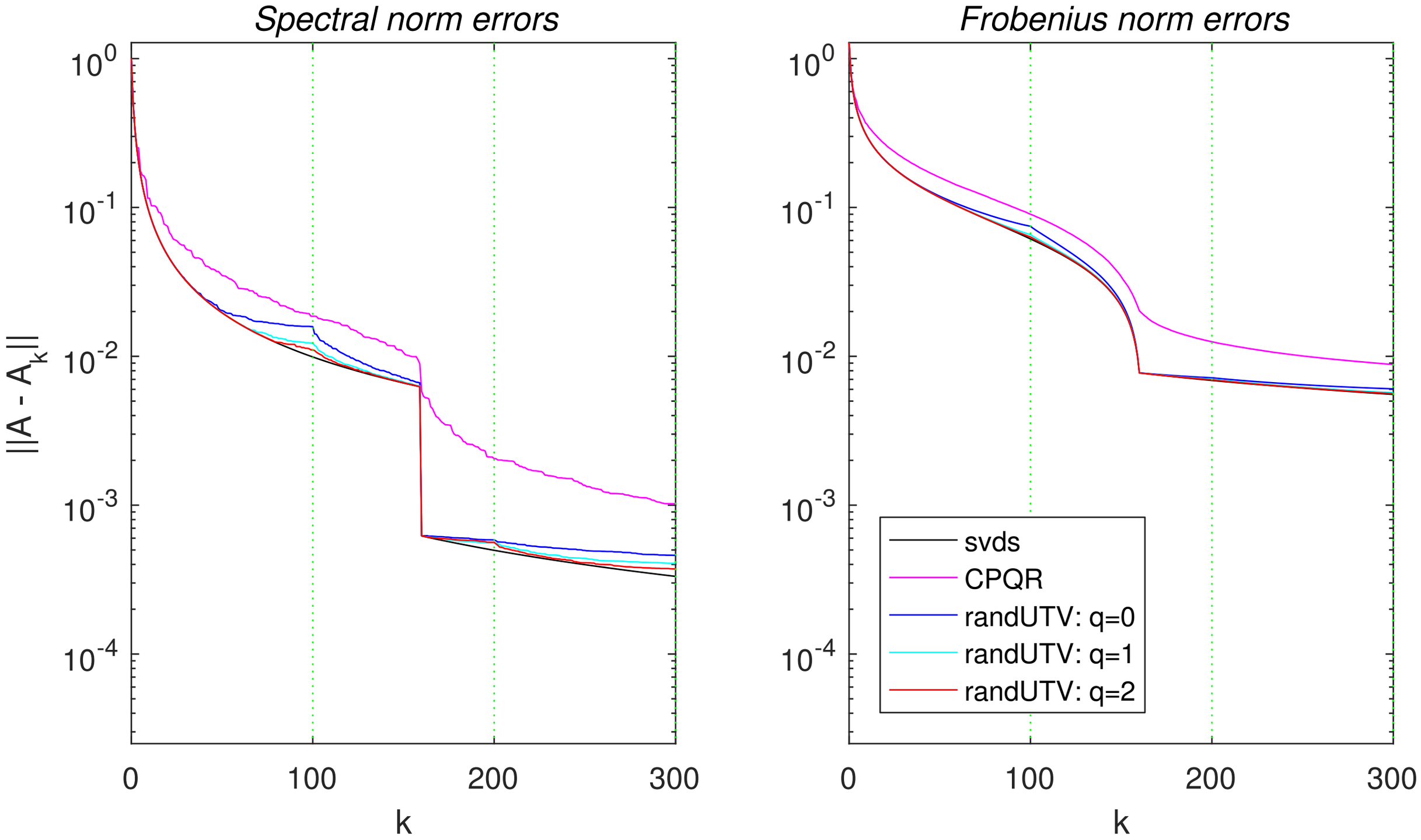
Our objective is in each step to find an approximation *to the linear subspace* spanned by the b dominant singular vectors of a matrix. The randomized range finder is perfect for this, especially when a small number of power iterations are performed. Easier and more natural than choosing pivoting vectors.



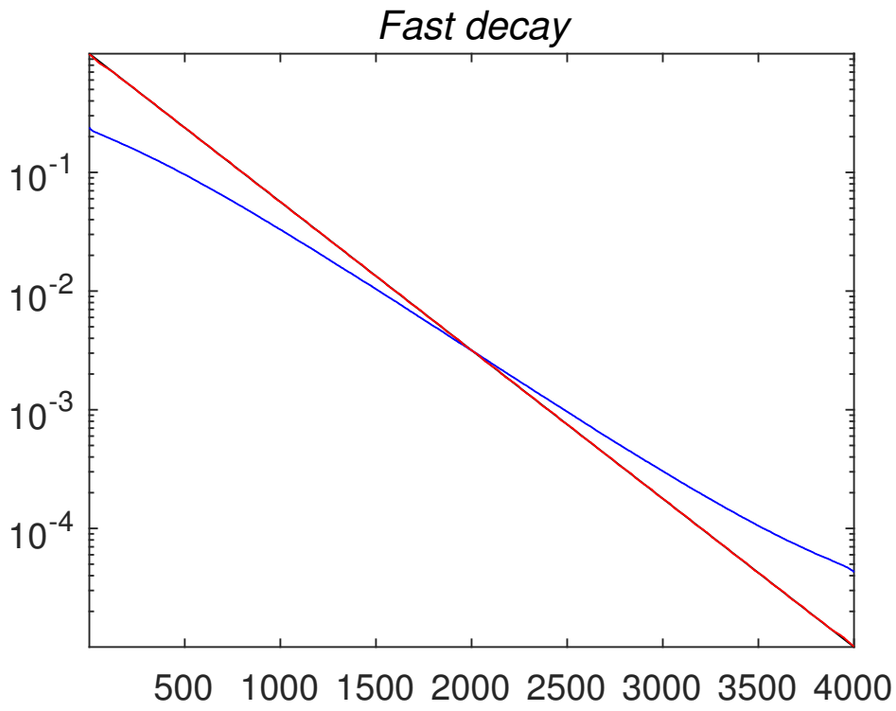
Rank-k approximation errors for the matrix "Fast Decay" of size 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*

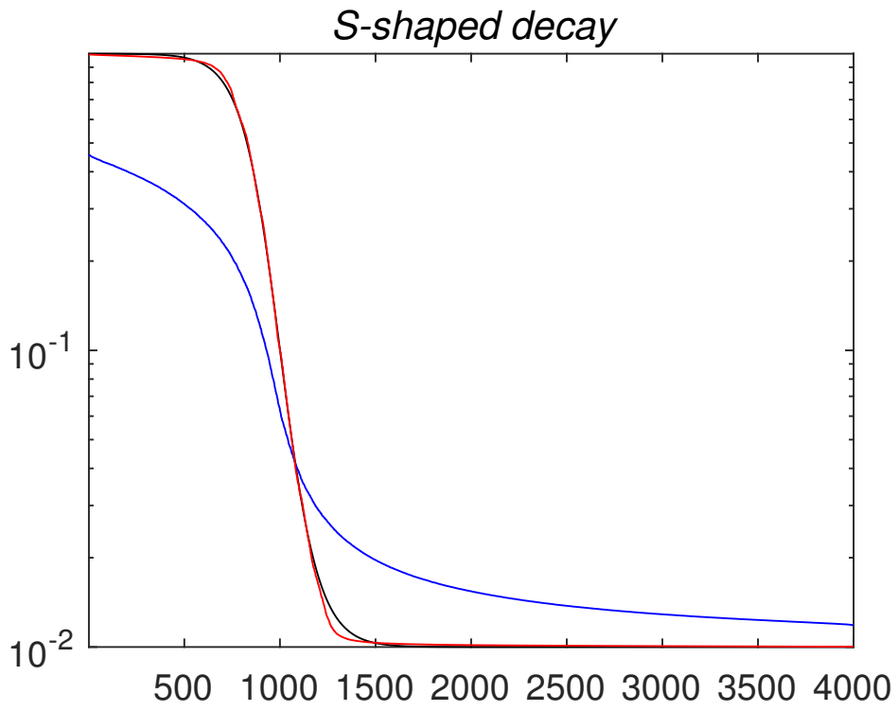


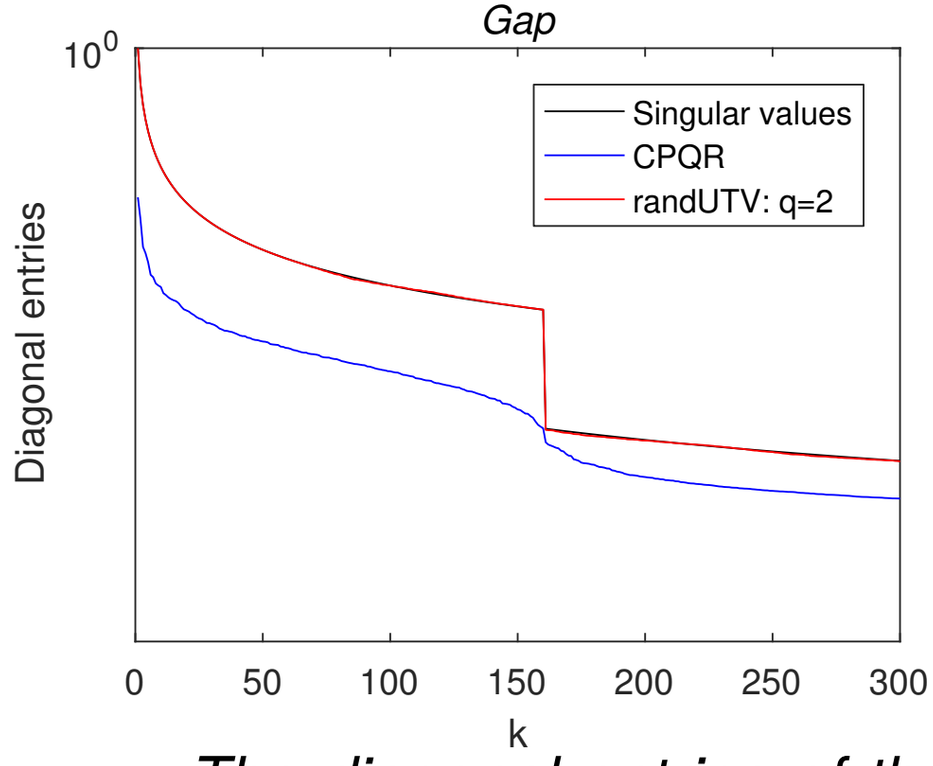
Rank-k approximation errors for the matrix "BIE" of size 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*

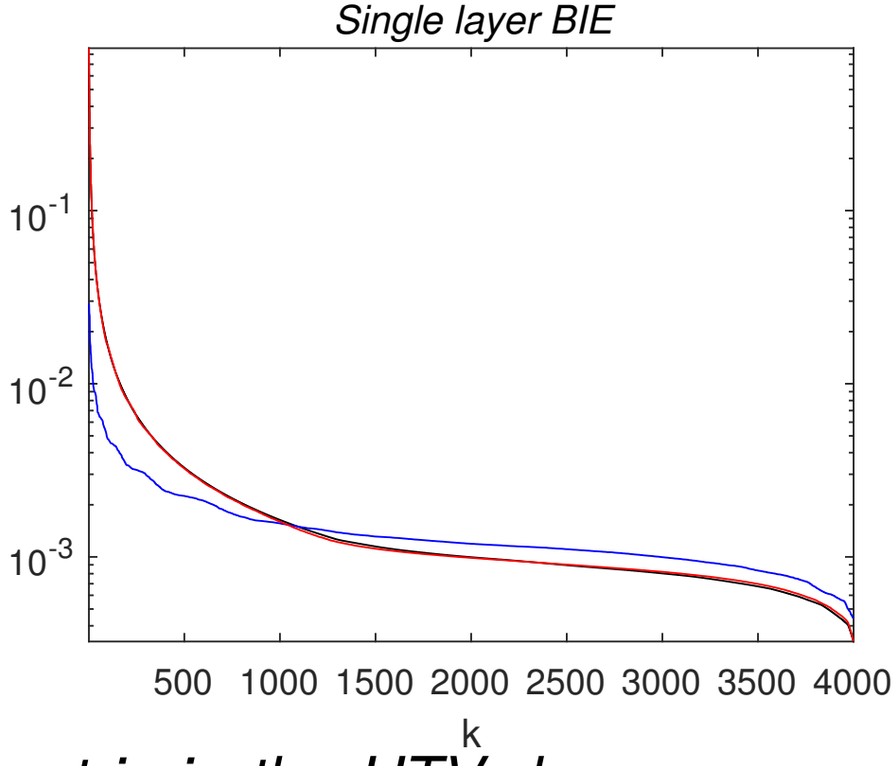


Rank-k approximation errors for $k \leq 300$ *for the matrix "Gap" of size* 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*





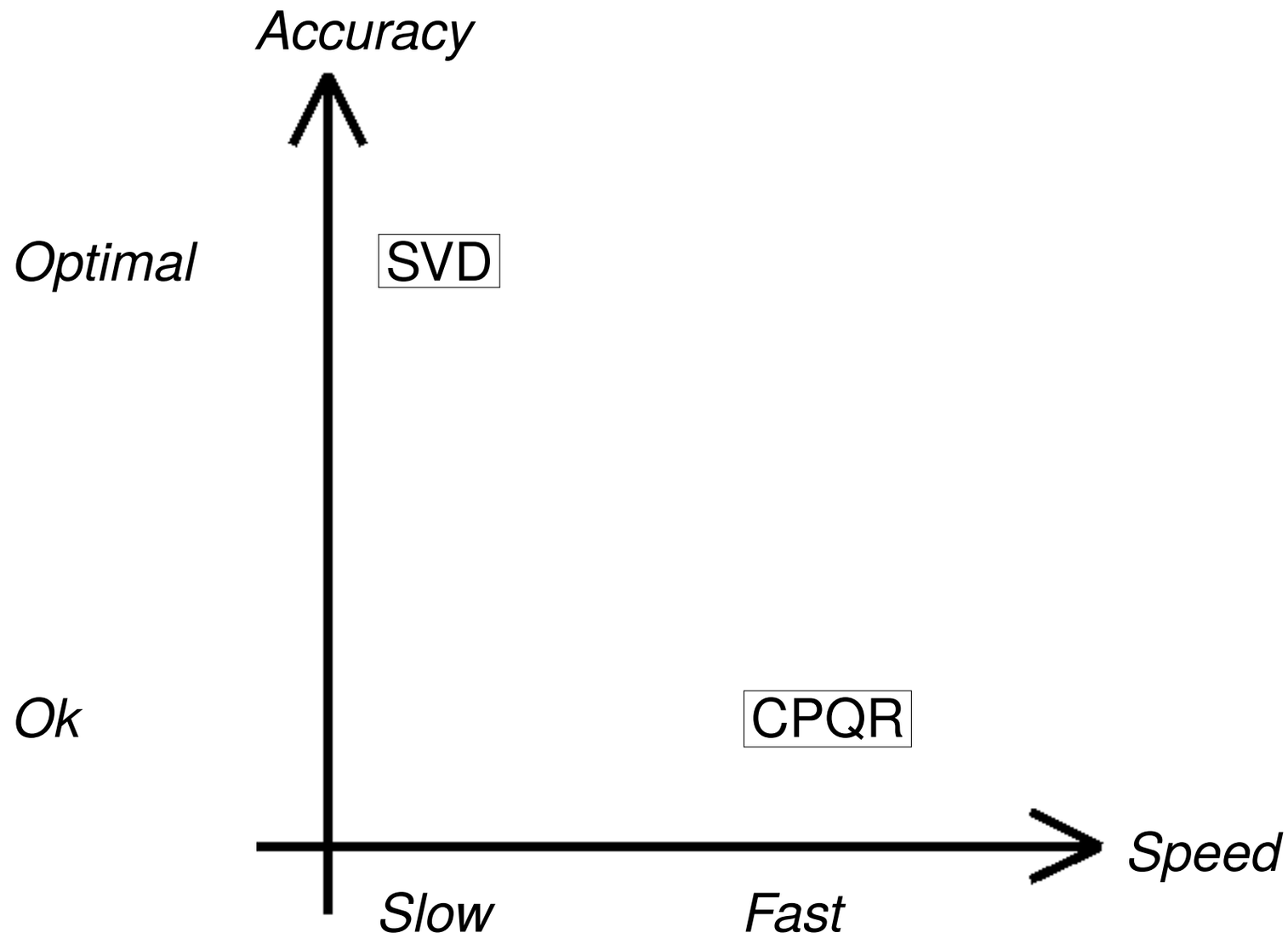




The diagonal entries of the **T***-matrix in the UTV decomposition (red) provide excellent approximations to the true singular values (black).*

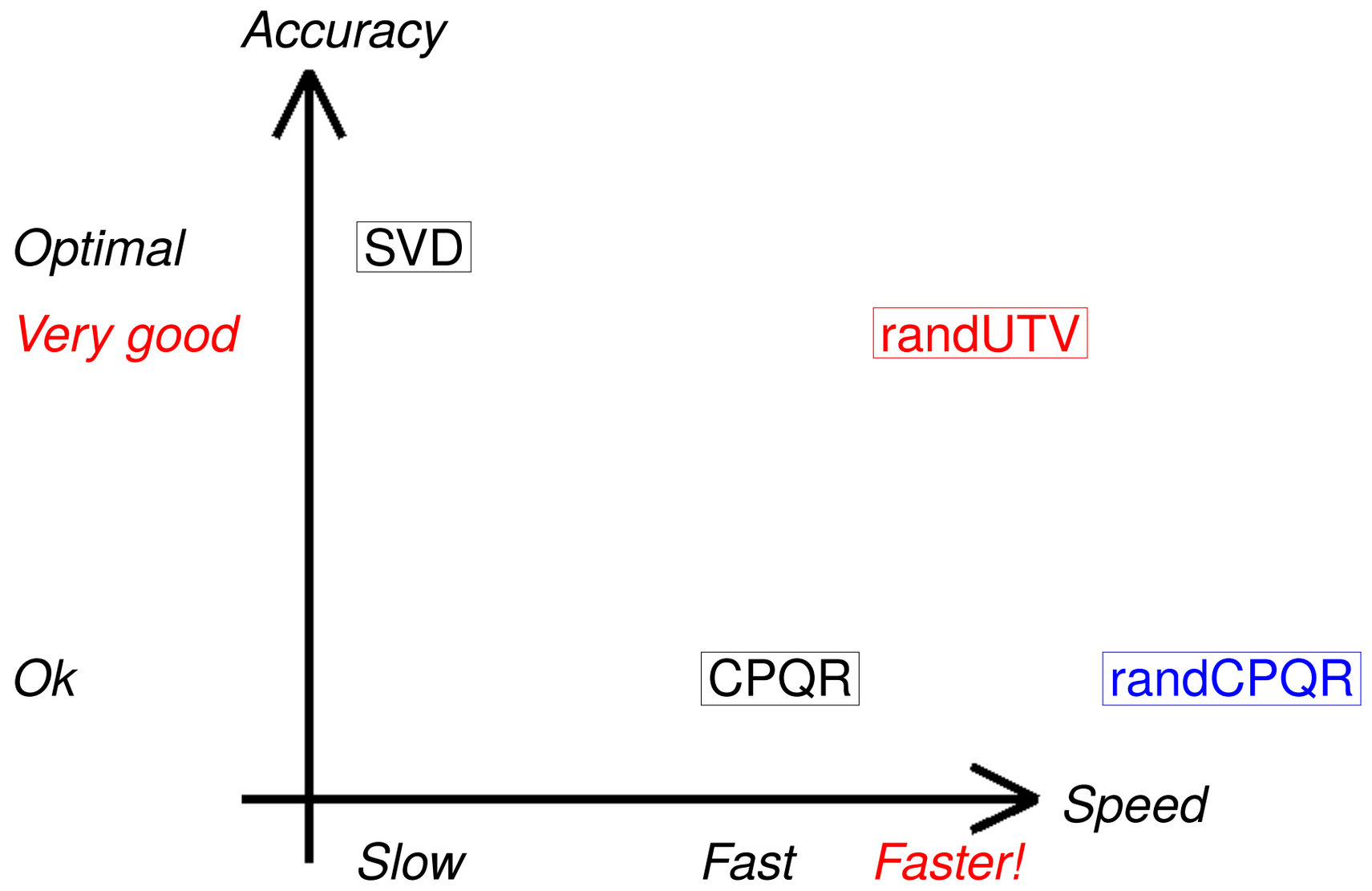
Computing full (or nearly full) rank revealing factorizations of matrices

For the task of computing low-rank approximations to matrices, the classical choice is between SVD and column pivoted QR (CPQR). SVD is slow, and CPQR is inaccurate:



Computing full (or nearly full) rank revealing factorizations of matrices

For the task of computing low-rank approximations to matrices, the classical choice is between SVD and column pivoted QR (CPQR). SVD is slow, and CPQR is inaccurate:



The randomized algorithm `randUTV` combines the best properties of both factorizations. Additionally, `randUTV` parallelizes better, and allows the computation of partial factorizations (like CPQR, but unlike SVD).

Computing full factorizations of matrices — Strassen type methods

The essential feature of the randomized methods described is that they enable us to expend almost all flops on the matrix-matrix computation, which is much faster per flop than other matrix operations.

Alternatively, use *asymptotically* faster methods for the matrix-matrix multiplication:

- Strassen: $O(n^{2.83})$. Stable. Reasonable breakeven point.
- Coppersmith-Winograd etc.: $O(n^{2.37})$. Unstable. Unreasonable breakeven point.

Observation:

Randomization allows you to use "fast" matrix-matrix multiplication algorithms to compute rank-revealing factorizations in a numerically stable way. In particular:

fast+stable matrix-matrix multiplication $\Rightarrow$ fast+stable linear system solve

Original work: Demmel, Dumitriu, and Holtz; Num. Math., **108**, 2007.

Computing full factorizations of matrices

Randomized scrambling:

- D.S. Parker, *Random butterfly transformations with applications in computational linear algebra*, Technical Report CSD-950023, UCLA, 1995.
- D. Lê, D.S. Parker, *Using randomization to make recursive matrix algorithms practical*, Journal of Functional Programming, **9**(6), 1999.
- J. Demmel, I. Dumitriu and O. Holtz, *Fast linear algebra is stable*, Numerische Mathematik, **108**(1), 2007.
- J. Demmel, L. Grigori, A. Rusciano, *An improved analysis and unified perspective on deterministic and randomized low rank matrix approximations*, arxiv #1910.00223, 2019.
- A. Gopal, P.G. Martinsson, *The PowerURV algorithm for computing rank-revealing full factorizations*, arXiv #1812.06007, 2018.

Randomized CPQR and UTV factorizations:

- P.G. Martinsson, G. Quintana-Orti, N. Heavner, and R. van de Geijn, *Householder QR Factorization With Randomization for Column Pivoting (HQRRP)*, SIAM Journal on Scientific Computation, **39**(2), 2017. *arxiv #1505.08115, May 2015.*
- J.A. Duersch, M. Gu, *Randomized QR with column pivoting*, SIAM Journal on Scientific Computing, **39**(4), C263–C291, 2017. *arXiv #1509.06820, Sep. 2015.*
- P.G. Martinsson, G. Quintana-Ortí, N. Heavner, *randUTV: A blocked randomized algorithm for computing a rank-revealing UTV factorization*, ACM TOMS, **45**(1), 2019.

Outline of talk

1. Introduction to randomized low rank approximation. *Done!*
2. Interpolatory and CUR factorizations (very brief). *Done!*
3. Rank revealing factorizations for matrices of full or nearly full rank. *Done!*
4. Brief survey of related research areas:
   - ◇ Structured random matrices.
   - ◇ Single-view ("streaming") algorithms.
   - ◇ Randomized block Krylov methods.
   - ◇ Approximation of kernel matrices — $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
   - ◇ (Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$.)

Research snapshots: Structured random matrices

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 $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

For a general dense matrix $\mathbf{A}$, the cost is $O(mnk)$, due to steps (2) and (5).

Research snapshots: Structured random matrices

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 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

For a general dense matrix $\mathbf{A}$, the cost is $O(mnk)$, due to steps (2) and (5).

These steps can be accelerated to $O(mn \log k)$ by using a *structured* random matrix $\mathbf{\Omega}$ (aka "fast Johnson-Lindenstrauss" transform). For instance:

- Subsampled random Fourier Transform (SRFT). Randomized Hadamard. Etc.
- Random *sparse* matrices. Can be surprisingly sparse!
- Random chains of Given rotations.

In theory, these methods come with much weaker performance guarantees.

In practice, the "good" transforms perform as well as Gaussians.

Certificate of accuracy: You can incorporate a small amount of Gaussian sampling to compute an error estimator. This removes the risk of using exotic and poorly understood random maps, *without changing the asymptotic cost.*

Research snapshots: Single view ("streaming") algorithms

Suppose you are given the following task:

- You seek an approximate rank-k factorization of an $m \times n$ matrix $\mathbf{A}$.
- You are allowed to see each entry of $\mathbf{A}$ *only once*. (Too large to store.)
- You cannot specify the order in which you see the elements.

Research snapshots: Single view ("streaming") algorithms

Suppose you are given the following task:

- You seek an approximate rank-k factorization of an $m \times n$ matrix $\mathbf{A}$.
- You are allowed to see each entry of $\mathbf{A}$ *only once*. (Too large to store.)
- You cannot specify the order in which you see the elements.

Solvable using randomized methods. (Only?) One option is the following:

- Fix oversampling parameters ℓ_{col} and ℓ_{row}, where $\ell_{\mathrm{col}}, \ell_{\mathrm{row}} \sim k$.
- Draw random matrices $\mathbf{\Omega}_{\mathrm{col}} \in \mathbb{R}^{n \times \ell_{\mathrm{col}}}$ and $\mathbf{\Omega}_{\mathrm{row}} \in \mathbb{R}^{\ell_{\mathrm{row}} \times m}$.
- As the matrix is streamed by you, incrementally build sample matrices

$$\mathbf{Y}_{\mathrm{col}} = \mathbf{A}\mathbf{\Omega}_{\mathrm{col}} \qquad \text{and} \qquad \mathbf{Y}_{\mathrm{row}} = \mathbf{\Omega}_{\mathrm{row}}\mathbf{A}.$$

- Compute the factorization from the information in $\{\mathbf{\Omega}_{\mathrm{col}}, \mathbf{Y}_{\mathrm{col}}, \mathbf{\Omega}_{\mathrm{row}}, \mathbf{Y}_{\mathrm{row}}\}$.

Research snapshots: Single view ("streaming") algorithms

Suppose you are given the following task:

- You seek an approximate rank-k factorization of an $m \times n$ matrix $\mathbf{A}$.
- You are allowed to see each entry of $\mathbf{A}$ *only once*. (Too large to store.)
- You cannot specify the order in which you see the elements.

Solvable using randomized methods. (Only?) One option is the following:

- Fix oversampling parameters ℓ_{col} and ℓ_{row}, where $\ell_{\text{col}}, \ell_{\text{row}} \sim k$.
- Draw random matrices $\boldsymbol{\Omega}_{\text{col}} \in \mathbb{R}^{n \times \ell_{\text{col}}}$ and $\boldsymbol{\Omega}_{\text{row}} \in \mathbb{R}^{\ell_{\text{row}} \times m}$.
- As the matrix is streamed by you, incrementally build sample matrices

$$\mathbf{Y}_{\text{col}} = \mathbf{A}\boldsymbol{\Omega}_{\text{col}} \qquad \text{and} \qquad \mathbf{Y}_{\text{row}} = \boldsymbol{\Omega}_{\text{row}}\mathbf{A}.$$

- Compute the factorization from the information in $\{\boldsymbol{\Omega}_{\text{col}}, \mathbf{Y}_{\text{col}}, \boldsymbol{\Omega}_{\text{row}}, \mathbf{Y}_{\text{row}}\}$.

Remarks:

- How to choose ℓ_{col} and ℓ_{row} is not well understood.
- Must have an estimate or bound for the numerical rank in advance.
- Typically lower accuracy than standard RSVD. (Do *not* use for out-of-core!)
- Tropp has proposed extracting additional samples ("core sample").

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 a 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 a 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; …

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.

Research snapshots: Approximation of kernel matrices

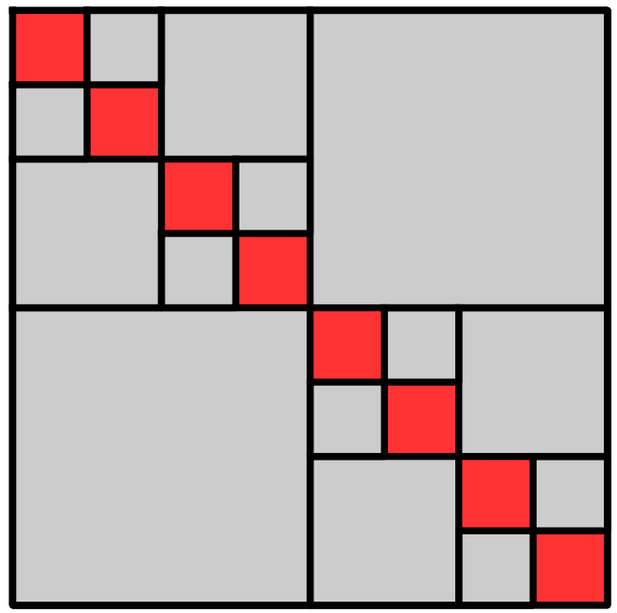
Consider a matrix of the form $\mathbf{A}(i,j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some kernel k and some set of points or vectors $\{\boldsymbol{x}_i\}_{i=1}^N$. (*Very* loose definition . . .)

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.



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.

Research snapshots: Solving $\mathbf{Ax} = \mathbf{b}$

Vast area of research!

Luckily for me, this was covered in Petros' talk.

Key points:

- Randomized low-rank approximation ("randomized SVD").
  - Superior performance in many regards, in particular for very large problems.
  - For a fixed number of matrix-vector multiplies, Krylov methods are more accurate.

- Essential benefit of randomization in linear algebra: *Reduces communication.*
  - Enables processing of huge data sets. (Out-of-core / streaming / cloud computing / ...)
  - Very fast on GPUs, distributed memory machines, etc.

- Two distinct paradigms for computing randomized approximations to matrices:
  1. Compute sketch via randomized *embedding*, involving all matrix entries.
     Very robust and reliable. Failure risk can be 10^{-10} or smaller. Not feasible in some environments.
  2. Compute sketch via randomized *sampling*.
     Cost can be less than cost for matvec. Very popular in "big data" applications where randomized sampling often serves as an enabling technology.

- In many situations, you can explicitly compute (or estimate) the residual error.
  "Certificate of accuracy" is especially useful for fast J-L transforms.

- Postdoc position is available at UT Austin!

Surveys

Adapted from Petros' slides!

- P. G. Martinsson and J. A. Tropp, *Randomized Numerical Linear Algebra: Foundations & Algorithms*, Acta Numerica, 2020.
- P. Drineas and M. W. Mahoney, *Lectures on Randomized Numerical Linear Algebra*. In the 2018 book *The Mathematics of Data,* published by AMS.
- P.G. Martinsson, *Randomized Methods for Matrix Computations*. In the 2018 book *The Mathematics of Data,* published by AMS.
- M. W. Mahoney and P. Drineas, *RandNLA: Randomized Numerical Linear Algebra*, Communications of the ACM, 2016.
- 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.
- N. Halko, P. G. Martinsson, J. A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, 2011.