Randomized algorithms for pivoting and for computing interpolatory and CUR factorizations

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. Efficient algorithms for computing CUR and interpolatory decompositions.

2. Randomized algorithms for computing *full* factorizations of matrices.
   Column pivoted QR in particular.

Interpolative and CUR decompositions

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

The *CUR* or *skeleton* approximation of $\mathbf{A}$ takes the form

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{C}} \; \underset{k \times k}{\mathbf{U}} \; \underset{k \times n}{\mathbf{R}}, \tag{1}$$

where $\mathbf{C}$ holds a subset of the columns of $\mathbf{A}$, and $\mathbf{R}$ holds a subset of the rows of $\mathbf{A}$. In other words, $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$ and $\mathbf{R} = \mathbf{A}(I_{\mathrm{S}}, :)$ for some index vectors J_{S} and I_{S}.

Closely related are the *interpolatory decompositions*

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{C}} \; \underset{k \times n}{\mathbf{Z}} \qquad \underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{X}} \; \underset{k \times n}{\mathbf{R}} \qquad \underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{X}} \; \underset{k \times k}{\mathbf{A}_{\mathrm{S}}} \; \underset{k \times n}{\mathbf{Z}}$$

where $\mathbf{C}$ and $\mathbf{R}$ are as in (1), and where $\mathbf{A}_{\mathrm{S}} = \mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}})$.

The fact that the matrices $\mathbf{R}$, $\mathbf{C}$, and $\mathbf{A}_{\mathrm{S}}$ are submatrices of $\mathbf{A}$ has important advantages:

- If $\mathbf{A}$ is sparse, then the factors $\mathbf{R}$ and $\mathbf{C}$ are sparse.
- If $\mathbf{A}$ is non-negative, then the factors $\mathbf{R}$, $\mathbf{C}$, and $\mathbf{A}_{\mathrm{S}}$ are non-negative.
- The factorizations allow for data interpretation.
- Reduced storage, since we can store the index vectors rather than the factors.
- Invaluable in the context of modern Fast Multipole Methods and Fast Direct Solvers.

Special case: Exact rank deficiency. Suppose $\mathbf{A}$ is of size $m \times n$ and *exact* rank k.

Let I and J be permutations or the row and column indices, and split $\mathbf{A}$ into four parts

$$\mathbf{A}(I, J) = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix},$$

so that $\mathbf{A}_{11}$ is the leading $k \times k$ submatrix. If $\mathbf{A}_{11}$ is non-singular, then necessarily

$$\mathbf{A}_{22} = \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}. \tag{2}$$

The *skeleton factorization* (CUR) follows directly from (2):

$$\mathbf{A}(I, J) = \underbrace{\begin{bmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{bmatrix}}_{=\mathbf{C}} \underbrace{\mathbf{A}_{11}^{-1}}_{=\mathbf{U}} \underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \end{bmatrix}}_{=\mathbf{R}}.$$

Analogously, the *interpolatory decomposition* takes the form

$$\mathbf{A}(I, J) = \underbrace{\begin{bmatrix} \mathbf{I} \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} \end{bmatrix}}_{=\mathbf{X}} \underbrace{\mathbf{A}_{11}}_{=\mathbf{A}_S} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{A}_{11}^{-1}\mathbf{A}_{12} \end{bmatrix}}_{=\mathbf{Z}}.$$

So existence of the CUR and the ID are straight-forward. But two questions arise:

(1) Are the factorizations well-conditioned?

(2) For a matrix of only *approximate* rank k, how close to optimal can you get?

Conditioning of the interpolatory decomposition

Recall the interpolatory decomposion:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}_{\mathrm{S}} & \mathbf{Z} \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{A}_{\mathrm{S}} = \mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}})$ is a $k \times$k submatrix of $\mathbf{A}$.

$\mathbf{X}$ and $\mathbf{Z}$ hold $k \times k$ identity matrices as submatrices. $\quad \Rightarrow \quad$ $\sigma_{\min}(\mathbf{X}) = \sigma_{\min}(\mathbf{Z}) = 1$

So $\mathbf{X}$ and $\mathbf{Z}$ are well-conditioned iff their *maximal* singular values are controlled.

Claim: Pick I_{S} and J_{S} so that $|\det(\mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}}))|$ is maximized over all $k \times k$ submatrices.

Then

$$(3) \qquad \sup_{i,j} |\mathbf{X}(i,j)| \leq 1 \qquad \text{and} \qquad \sup_{i,j} |\mathbf{Z}(i,j)| \leq 1.$$

Proof: Use Cramer's rule to bound entries of $\mathbf{A}_{21}\mathbf{A}_{11}^{-1}$ and $\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$. □

The bounds (3) imply that $\kappa(\mathbf{X}) \leq \sqrt{1 + k(m-k)}$ and $\kappa(\mathbf{Z}) \leq \sqrt{1 + k(n-k)}$.

So YES, a well-conditioned ID exists. (Finding it is another matter . . .)

Conditioning of the interpolatory decomposition

Recall the interpolatory decomposion:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}_{\mathrm{S}} & \mathbf{Z} \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{A}_{\mathrm{S}} = \mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}})$ is a $k \times$k submatrix of $\mathbf{A}$.

$\mathbf{X}$ and $\mathbf{Z}$ hold $k \times k$ identity matrices as submatrices. $\quad \Rightarrow \quad$ $\sigma_{\min}(\mathbf{X}) = \sigma_{\min}(\mathbf{Z}) = 1$

So $\mathbf{X}$ and $\mathbf{Z}$ are well-conditioned iff their *maximal* singular values are controlled.

Claim: Pick I_{S} and J_{S} so that $|\det(\mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}}))|$ is maximized over all $k \times k$ submatrices.

Then

(3) $$\sup_{i,j} |\mathbf{X}(i,j)| \leq 1 \qquad \text{and} \qquad \sup_{i,j} |\mathbf{Z}(i,j)| \leq 1.$$

Proof: Use Cramer's rule to bound entries of $\mathbf{A}_{21}\mathbf{A}_{11}^{-1}$ and $\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$. □

The bounds (3) imply that $\kappa(\mathbf{X}) \leq \sqrt{1 + k(m-k)}$ and $\kappa(\mathbf{Z}) \leq \sqrt{1 + k(n-k)}$.

So YES, a well-conditioned ID exists. (Finding it is another matter . . .)

What about CUR?

Conditioning of the interpolatory decomposition

Recall the interpolatory decomposion:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}_{\mathrm{S}} & \mathbf{Z} \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{A}_{\mathrm{S}} = \mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}})$ is a $k \times$k submatrix of $\mathbf{A}$.

$\mathbf{X}$ and $\mathbf{Z}$ hold $k \times k$ identity matrices as submatrices. $\quad \Rightarrow \quad$ $\sigma_{\min}(\mathbf{X}) = \sigma_{\min}(\mathbf{Z}) = 1$

So $\mathbf{X}$ and $\mathbf{Z}$ are well-conditioned iff their *maximal* singular values are controlled.

Claim: Pick I_{S} and J_{S} so that $|\det(\mathbf{A}(I_{\mathrm{S}}, J_{\mathrm{S}}))|$ is maximized over all $k \times k$ submatrices.

Then

$$\text{(3)} \qquad \sup_{i,j} |\mathbf{X}(i,j)| \leq 1 \qquad \text{and} \qquad \sup_{i,j} |\mathbf{Z}(i,j)| \leq 1.$$

Proof: Use Cramer's rule to bound entries of $\mathbf{A}_{21}\mathbf{A}_{11}^{-1}$ and $\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$. □

The bounds (3) imply that $\kappa(\mathbf{X}) \leq \sqrt{1 + k(m-k)}$ and $\kappa(\mathbf{Z}) \leq \sqrt{1 + k(n-k)}$.

So YES, a well-conditioned ID exists. (Finding it is another matter . . .)

What about CUR? It is in general *not* well conditioned.

Optimality in terms of low rank approximation

Recall Eckart–Young theorem: $\inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\} = \sigma_{k+1}(\mathbf{A})$

(Our default is that $\|\cdot\|$ refers to the ℓ^2 operator norm.)

Question: How close to optimal can you get with an ID or a CUR decomposition?

Optimality in terms of low rank approximation

Recall Eckart–Young theorem: $\inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\} = \sigma_{k+1}(\mathbf{A})$

(Our default is that $\|\cdot\|$ refers to the ℓ^2 operator norm.)

Question: How close to optimal can you get with an ID or a CUR decomposition?

Very well studied subject.

Tends to be within factor $\sim \sqrt{kn}$ of optimal in the worst case.

Often much better in practice, in particular when the singular values decay rapidly.

Theorem: Let $\mathbf{A}$ be an $m \times n$ matrix, and let $k < \min(m, n)$. There exists a matrix $\mathbf{U}$ of size $k \times k$, and index vector I_{S} and J_{S} of length k such that

$$\|\mathbf{A} - \mathbf{CUR}\| \leq \big(1 + 2\sqrt{k}(\sqrt{m} + \sqrt{n})\big)\sigma_{k+1}(\mathbf{A}),$$

where

$$\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}}), \qquad \mathbf{R} = \mathbf{A}(I_{\mathrm{S}}, :).$$

References: Goreinov, S.A., Tyrtyshnikov, E.E., Zamarashkin, N.L. (1997); Goreinov, S.A., Tyrtyshnikov, E.E. (2001). Survey: Ballani, J. & Kressner, D. (2016).

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector J_{S} of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

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

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector J_{S} of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

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

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of $\mathbf{A}$. This results in a factorization

$$\underset{m \times n}{\mathbf{A}(:, J)} \approx \underset{m \times k}{\mathbf{Q}} \quad \begin{bmatrix} \underset{k \times k}{\mathbf{R}_{11}} & \underset{k \times (n-k)}{\mathbf{R}_{12}} \end{bmatrix}$$

Set $J_{\mathrm{S}} = J(1 : k)$, and pull out the factor $\mathbf{R}_{11}$ to form the column-ID:

$$\mathbf{A}(:, J) \approx \mathbf{Q}\mathbf{R}_{11} \quad \begin{bmatrix} \mathbf{I} & \mathbf{R}_{11}^{-1}\mathbf{R}_{12} \end{bmatrix} = \mathbf{A}(:, J_{\mathrm{S}})\, \mathbf{Z}(:, J).$$

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector J_{S} of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

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

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of $\mathbf{A}$. This results in a factorization

$$\underset{m \times n}{\mathbf{A}(:, J)} \approx \underset{m \times k}{\mathbf{Q}} \quad \begin{bmatrix} \underset{k \times k}{\mathbf{R}_{11}} & \underset{k \times (n-k)}{\mathbf{R}_{12}} \end{bmatrix}$$

Set $J_{\mathrm{S}} = J(1:k)$, and pull out the factor $\mathbf{R}_{11}$ to form the column-ID:

$$\mathbf{A}(:, J) \approx \mathbf{Q}\mathbf{R}_{11} \quad \begin{bmatrix} \mathbf{I} & \mathbf{R}_{11}^{-1}\mathbf{R}_{12} \end{bmatrix} = \mathbf{A}(:, J_{\mathrm{S}})\, \mathbf{Z}(:, J).$$

Notes:

- Orthonormality must be maintained *scrupulously*. Use Householder or "double" Gram-Schmidt.
- CPQR can in principle fail (e.g. the "Kahan counter example"), but in practice it works very well.
- Reasonably computationally efficient for matrices that fit in RAM.
- Sophisticated versions of CPQR have been developed that *guarantee* close to optimal column selection, as well as bounding all elements of $\mathbf{R}_{11}^{-1}\mathbf{R}_{12}$. *(Gu & Eisenstat SISC 1996)*

The column selection problem: How do you efficiently compute a CUR/ID?

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector J_{S} of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

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

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of $\mathbf{A}$. This results in a factorization

$$\underset{m \times n}{\mathbf{A}(:, J)} \approx \underset{m \times k}{\mathbf{Q}} \quad \begin{bmatrix} \underset{k \times k}{\mathbf{R}_{11}} & \underset{k \times (n-k)}{\mathbf{R}_{12}} \end{bmatrix}$$

Set $J_{\mathrm{S}} = J(1:k)$, and pull out the factor $\mathbf{R}_{11}$ to form the column-ID:

$$\mathbf{A}(:, J) \approx \mathbf{Q}\mathbf{R}_{11} \quad \begin{bmatrix} \mathbf{I} & \mathbf{R}_{11}^{-1}\mathbf{R}_{12} \end{bmatrix} = \mathbf{A}(:, J_{\mathrm{S}})\, \mathbf{Z}(:, J).$$

Questions:

- Can you efficiently process large matrices that do not fit in fast memory?
- Can you efficiently process huge *sparse* matrices?
- Can you improve on the practical speed of CPQR? Even on the $O(mnk)$ complexity?

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 $\mathbf{A}$ 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 $\mathbf{F}$, 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 $\mathbf{A}$:

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

Proof: Assume that

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

and that

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

Then

$$\mathbf{A}(:, J_{\mathrm{S}})\mathbf{Z} \overset{(2)}{=} \mathbf{EF}(:, J_{\mathrm{S}})\mathbf{Z} \overset{(3)}{=} \mathbf{EF} = \mathbf{A}.$$

The column selection problem — through a *sketch*

Simple theorem: Let $\mathbf{A}$ 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 $\mathbf{F}$, 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 $\mathbf{A}$:

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

Question: How do you find a $k \times n$ matrix $\mathbf{F}$ such that $\mathbf{A} = \mathbf{EF}$ for some $\mathbf{E}$?

The column selection problem — through a *sketch*

Simple theorem: Let $\mathbf{A}$ 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 $\mathbf{F}$, 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 $\mathbf{A}$:

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

Question: How do you find a $k \times n$ matrix $\mathbf{F}$ such that $\mathbf{A} = \mathbf{EF}$ for some $\mathbf{E}$?

Randomized embedding! Draw a $k \times m$ Gaussian random matrix $\mathbf{\Omega}$ and set

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

The probability that $\mathbf{A} = \mathbf{EF}$ for some $\mathbf{E}$ is 1. (Of course, $\mathbf{E} = \mathbf{AF}^{\dagger}$.)

We do not need to know the factor $\mathbf{E}$*! 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 $\boldsymbol{\Omega}$;

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

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

Basic version: Use a *Gaussian* random matrix.

Complexity is $O(mnk)$ for a general dense matrix. Very high *practical* speed.

Complexity is $O(\mathrm{nnz}(\mathbf{A})k)$ for a sparse matrix. Again, high *practical* speed.

In some ways optimal sampling. Well supported by theory, e.g.:

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

(Skeletonization slightly increases the error: $\|\mathbf{A} - \mathbf{A}(:, J_{\mathrm{S}})\mathbf{Z}\| \geq \|\mathbf{A} - \mathbf{A}\mathbf{F}^{\dagger}\mathbf{F}\|$.)

Note: The probability that a set of k columns is sampled is in a certain sense proportional to its spanning volume. This is precisely the property we are after.

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 $\boldsymbol{\Omega}$;

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

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

Variation: Use a Gaussian matrix $\boldsymbol{\Omega}$, and incorporate *power iteration*.

Replace $\mathbf{F} = \boldsymbol{\Omega}\mathbf{A}$ in step (2) by $\mathbf{F} = \boldsymbol{\Omega}\left(\mathbf{A}\mathbf{A}^*\right)^q\mathbf{A}$ for a small integer q. Say $q = 1$ or $q = 2$. (I.e. do classical subspace iteration.)

This makes row($\mathbf{F}$) better aligned with the space spanned by the dominant k right singular vectors of $\mathbf{A}$.

Enhances accuracy, at cost of more work. Strong supporting theory. E.g., with $p = k$:

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

Re-orthonormalization is sometimes required to avoid loss of accuracy due to round-off.

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 $\boldsymbol{\Omega}$;

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

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

Variation: Use a *structured random matrix* $\boldsymbol{\Omega}$. ("Fast Johnson-Lindenstrauss transform")

Idea: Use a matrix $\boldsymbol{\Omega}$ for which $\boldsymbol{\Omega}\mathbf{A}$ can be evaluated efficiently.

- Subsampled Randomized Fourier Transform (SRFT). Can be applied using FFT like methods. Cost is $O(mn\log(k))$ instead of $O(mnk)$.

- Sparse random matrix. Put only a couple of non-zero entries in each column. (Entries can be restricted to ± 1 for additional efficiency.) $O(mn)$ cost attainable.

Works quite well in practice. SRFTs are almost as good as Gaussians.
However, *far* weaker theory is available.

Cannot be combined with power iteration.

Ailon/Chazelle 2006; Liberty/Rokhlin/Tygert/Woolfe 2006; Halko/Martinsson/Tropp 2011; Clarkson/Woodruff 2013.

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 $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: Discrete empirical interpolation method (DEIM)

Idea: Use the sample matrix $\mathbf{F}$ to build an approximate truncated SVD of $\mathbf{A}$.
(Form SVD of $(\mathbf{AF}^{\dagger})\,\mathbf{F}$.) Requires one additional matrix-matrix multiplication.

Then perform partially pivoted LU on a thin matrix formed by the singular vectors to pick the spanning columns.

DEIM sometimes produces slightly more optimal column selection than CPQR.

DEIM can be faster than doing CPQR directly. (Which is perhaps counterintuitive!)

Often excellent choice.

Reference: Sorensen & Embree SISC 2016.

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 $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: "Poor man's DEIM"

Idea: Skip forming the SVD, and just do partially pivoted LU on $\mathbf{F}^*$ directly!

Very economical. Works about as well as either CPQR, or regular DEIM.

Becomes particularly accurate with one step of power iteration.

Inspired by work by Trefethen and Schreiber (SIMAX 1990) on Gaussian elimination on random matrices.

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 $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.

Variation: Other possibilities:

- Can use sophisticated methods à la Gu-Eisenstat RRQR in Step (3). Leads to methods that are well supported by theory. Little improvement in practice, however.
- Can use the sketch to form an SVD. Then estimate *leverage scores*, and draw the columns through randomized sampling on the index set $\{1, 2, \ldots, n\}$ using the resulting probability distribution. Rarely competitive in practice.
  (However, approaches of this type can be powerful for *huge* matrices where the matrix-vector multiplication is not accessible.)

Numerical experiments

Numerical experiments

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

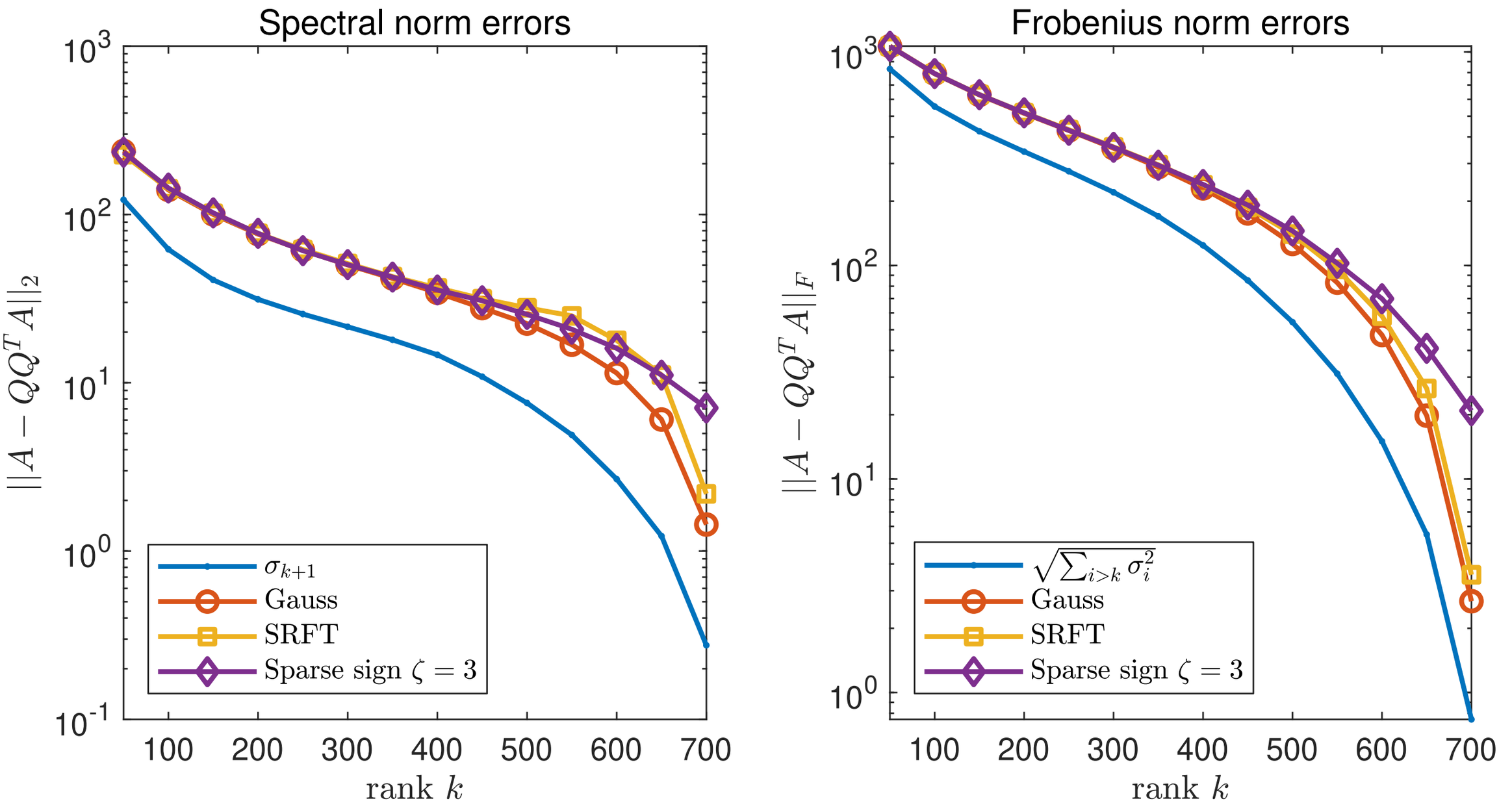
We will compare:

- Optimality: How good of a basis for the row 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.

(To be precise, in these experiments, we sketched the *column space*, not the row space.)

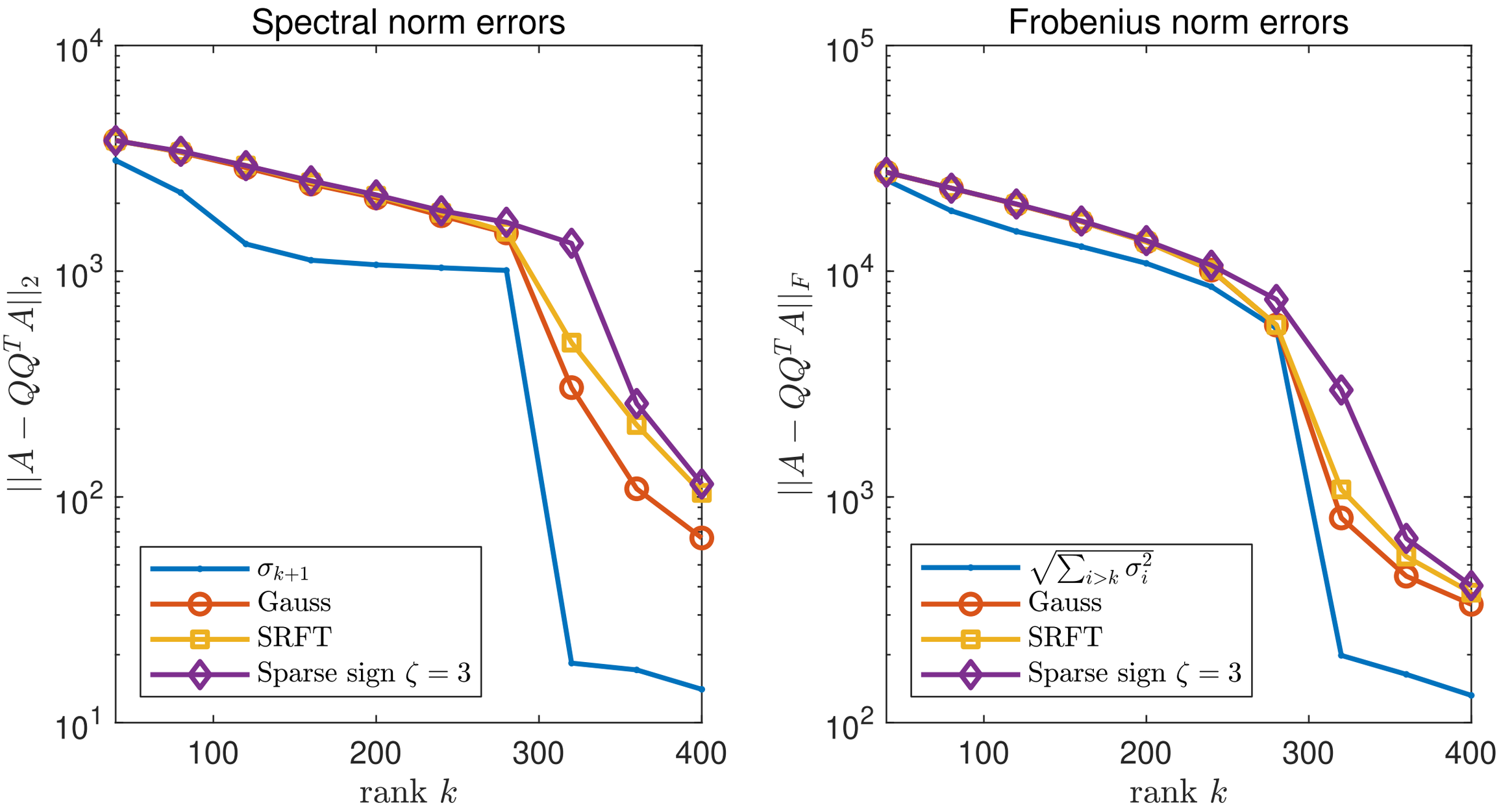


*The "*MNIST*" test matrix is dense and of size* 784 $\times$ 60 000 *where each column holds one hard drawn digit between 0 and 9. The matrix is* 80% *sparse.*

Comparison of different random matrices — accuracy

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

(To be precise, in these experiments, we sketched the *column space*, not the row space.)

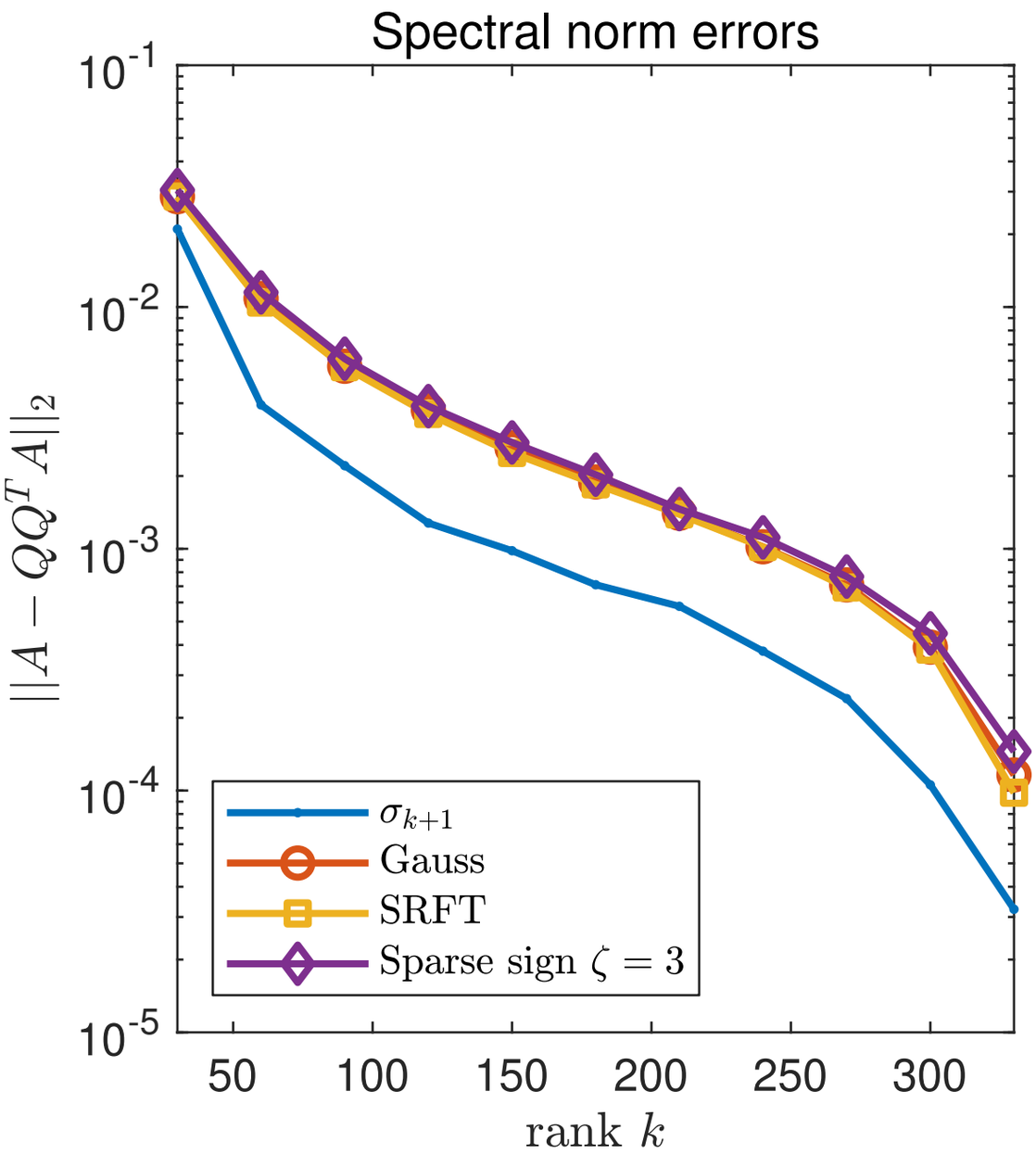


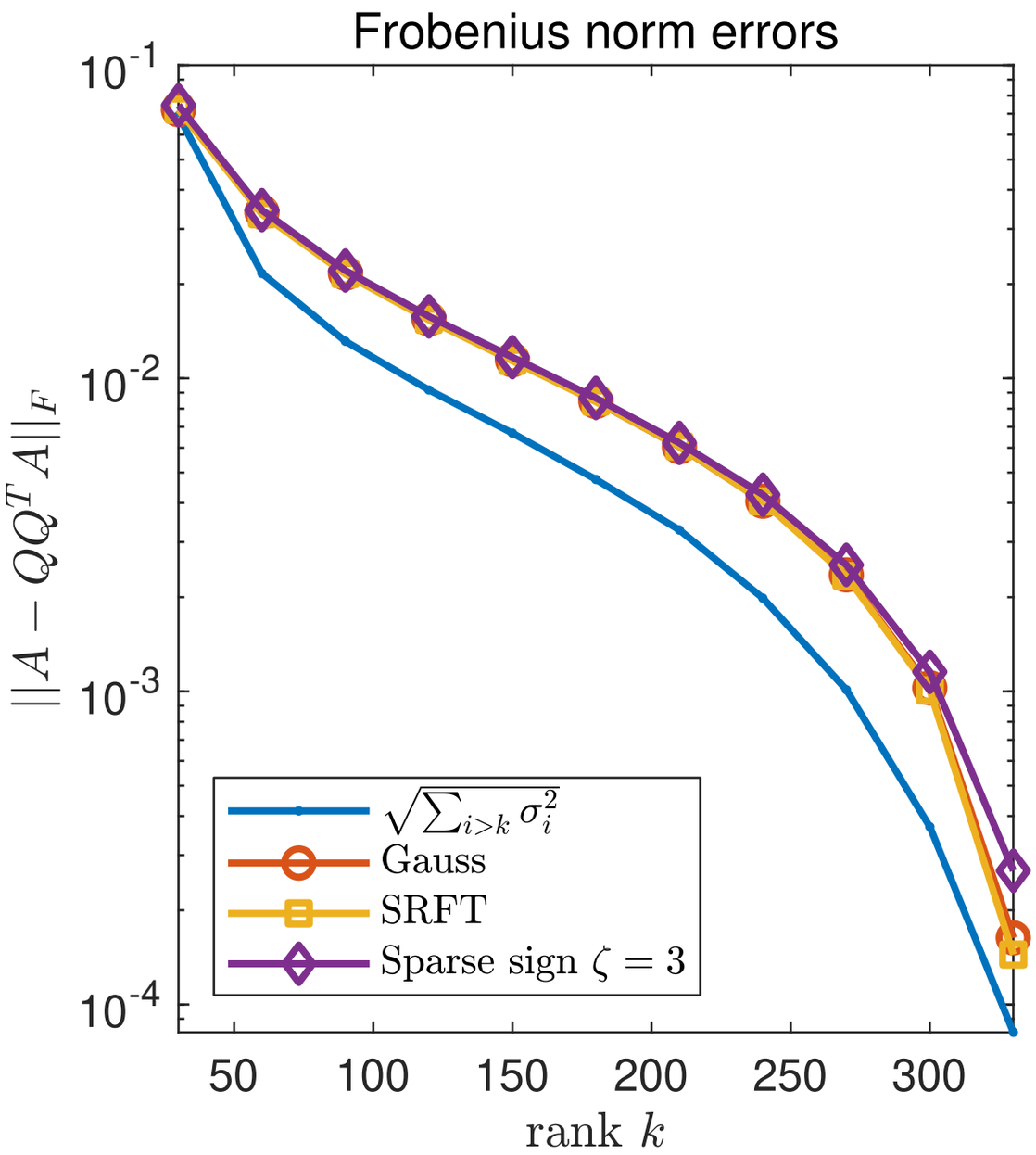
*The "*LARGE*" test matrix is taken from a linear programming example. It is sparse, of size* 4 282 × 8 617, *with* 20 635 *nonzero entries.*

Comparison of different random matrices — accuracy

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

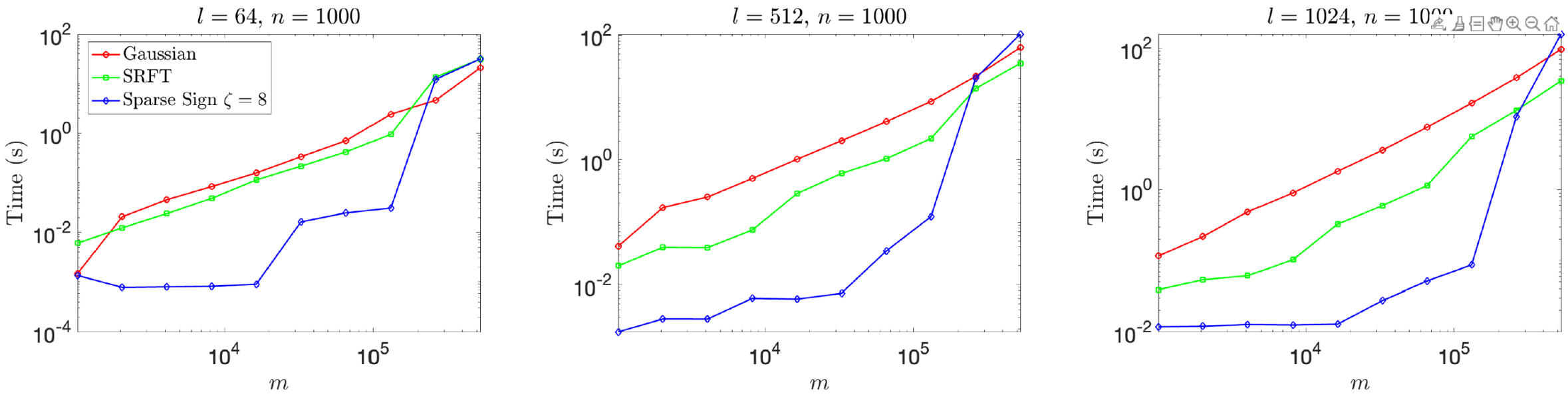
(To be precise, in these experiments, we sketched the *column space*, not the row space.)





*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* $n = 1000$*.*

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

Numerical experiments

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

We will compare:

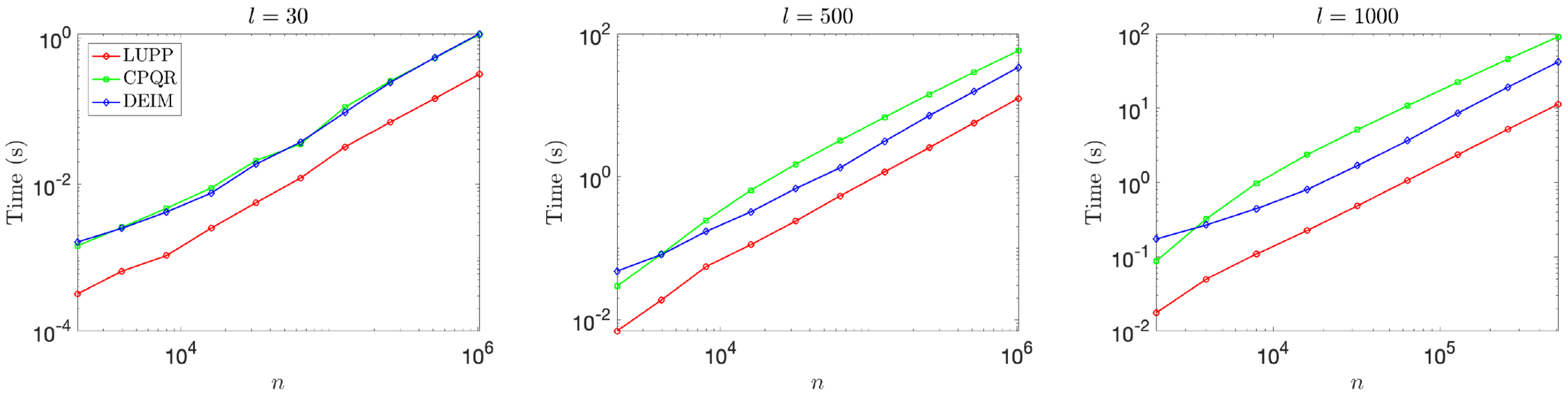
- Column pivoted QR
- DEIM: Form approximate SVD, then partially pivoted LU on the singular vectors.
- Partially pivoted LU directly on the sketching matrix. ("Poor man's DEIM")
- (Form approximate RSVD, then compute "leverage scores", then draw columns based on the leverage scores.)

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*.



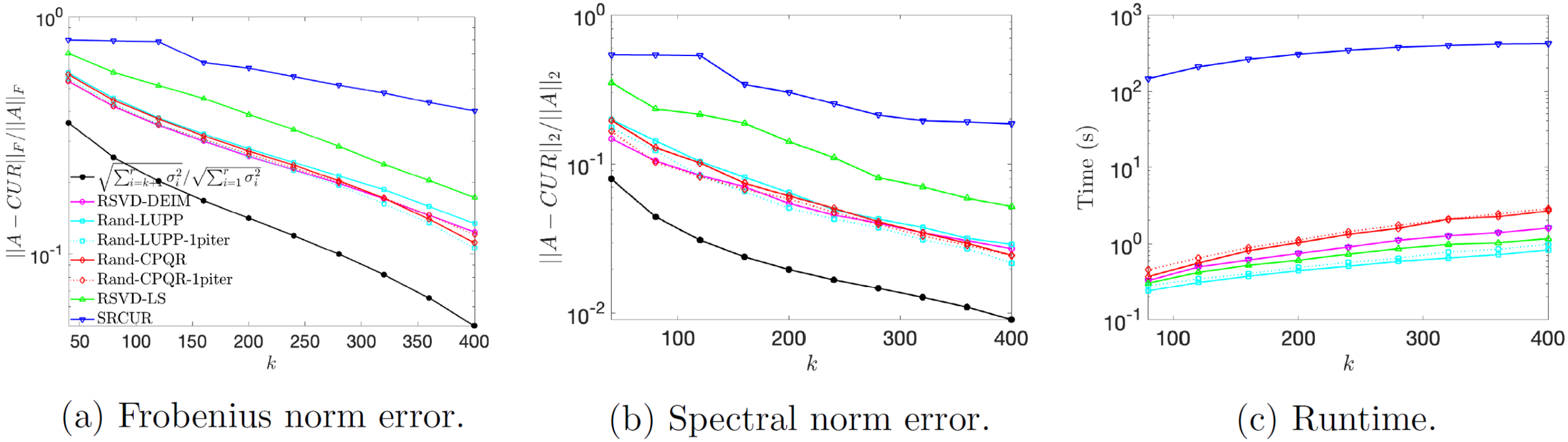
The runtime of various pivoting schemes on the sketches of size $n \times \ell$, scaled with respect to the problem size n, at different embedding dimension ℓ.

Observe that the dimension of the sketch is quite high in these 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*.



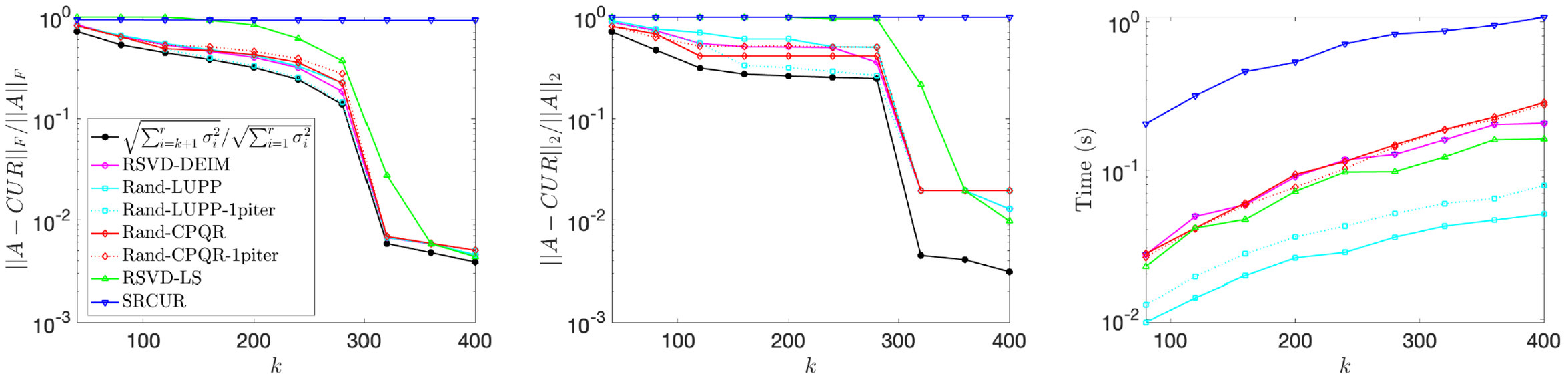
(a) Frobenius norm error. (b) Spectral norm error. (c) Runtime.

*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*.



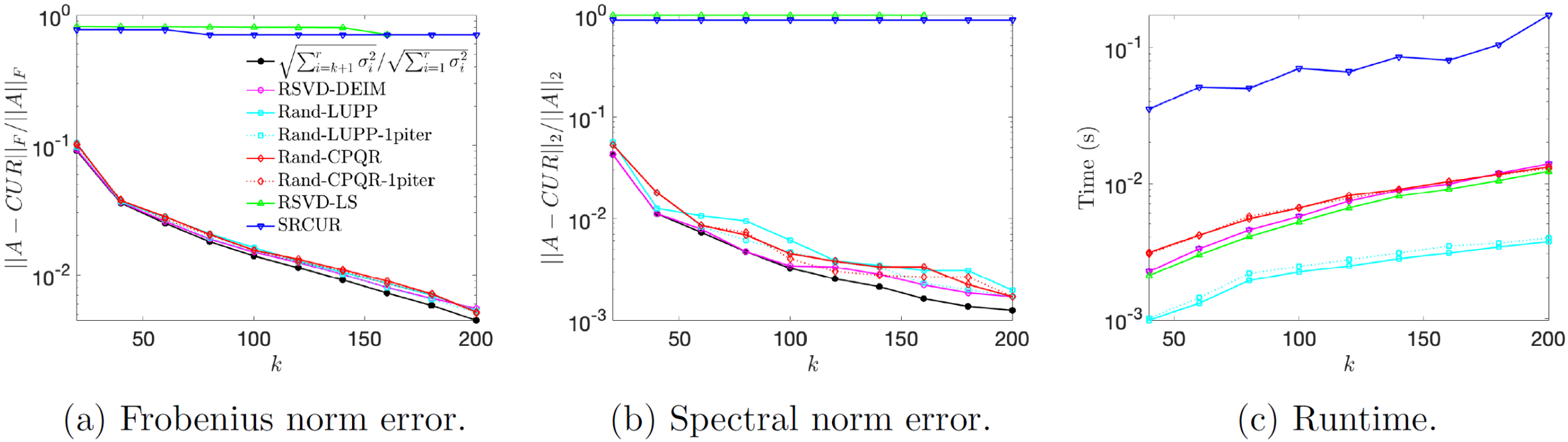
(a) Frobenius norm error. (b) Spectral norm error. (c) Runtime.

*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*.



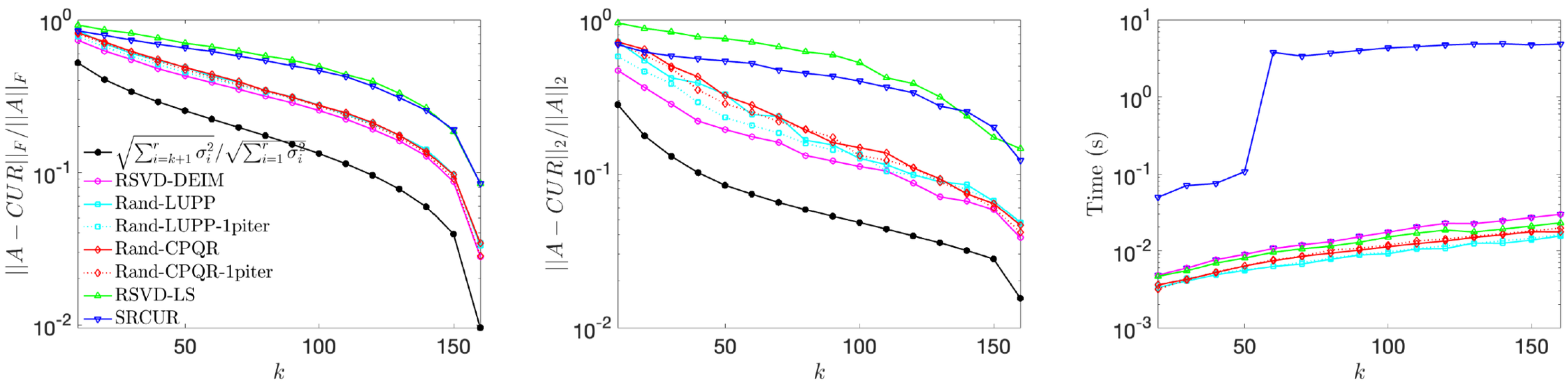
(a) Frobenius norm error. (b) Spectral norm error. (c) Runtime.

*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.*

Lessons from numerical experiments:

A bit tendentious, perhaps...

- 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.
- 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.

  They are about equally good at picking columns. DEIM perhaps slight winner.
  Partially pivoted LU ("Poor man's DEIM") is the fastest by a margin.
- (We could not get techniques based on leverage scores to perform competitively in this environment. Did we do something wrong? Advice welcome.)

Lessons from numerical experiments:

A bit tendentious, perhaps...

- 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.
- 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.

  They are about equally good at picking columns. DEIM perhaps slight winner.
  Partially pivoted LU ("Poor man's DEIM") is the fastest by a margin.
- (We could not get techniques based on leverage scores to perform competitively in this environment. Did we do something wrong? Advice welcome.)

Conclusion: Keep it simple!

Manuscript with full details forthcoming; joint work with Yijun Dong of UT-Austin.

Outline of talk

1. *[Done]* Algorithms for computing CUR and interpolatory decompositions.

2. Randomized algorithms for computing *full* factorizations of matrices.
   Column pivoted QR in particular.

Randomized algorithms for computing full factorizations of matrices

Let us consider some standard matrix decompositions:

• Unpivoted QR (QR)	• Column pivoted QR (CPQR)
• Partially pivoted LU	• Fully pivoted LU
	• SVD

Randomized algorithms for computing full factorizations of matrices

Let us consider some standard matrix decompositions:

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

Randomized algorithms for computing full factorizations of matrices

Let us consider some standard matrix decompositions:

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

The factorizations on the left can be *blocked*, which makes it possible to cast almost all flops in efficient matrix-matrix multiplications. They are “communication efficient”.

In contrast, 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).

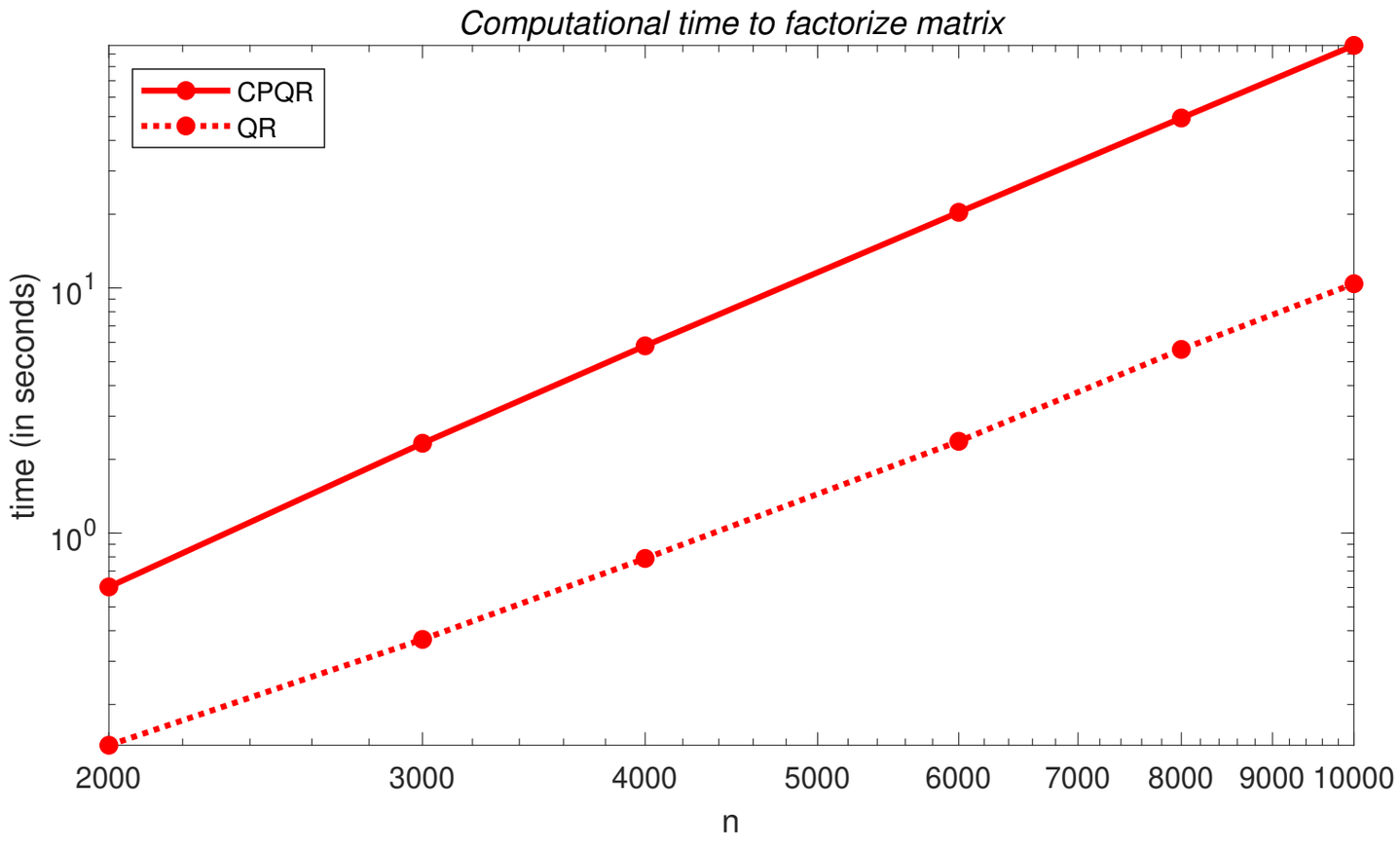
“BLAS3 on the left, BLAS2 on the right.”

Mea culpa: Our treatment is simplistic. Much progress has been made on communication efficient implementations of CPQR, SVD, etc. See L. Grigori E-NLA talk.

Randomized algorithms for computing full factorizations of matrices

Let us consider some standard matrix decompositions:

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



Matlab on a standard office desktop with an i7-6700k CPU, circa 2016.

Randomized algorithms for computing full factorizations of matrices

Let us consider some standard matrix decompositions:

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

Objective: Build rank-revealing algorithms with speed similar to unpivoted QR.

Note: All methods in the remainder of the talk have complexity $O(n^3)$.

Our objective is to improve the scaling factor.

(And to make them more rank revealing to boot!)

Randomized algorithms for computing full factorizations of matrices

The idea of using "randomized scrambling" to avoid having to slog through all the rank-one updates has a long history. For instance:

D. Stott Parker (1995): Can you forego pivoting when solving $\mathbf{Ax} = \mathbf{b}$ via Gaussian elimination, and still retain (some) stability?

Randomized algorithms for computing full factorizations of matrices

The idea of using "randomized scrambling" to avoid having to slog through all the rank-one updates has a long history. For instance:

D. Stott Parker (1995): Can you forego pivoting when solving $\mathbf{Ax} = \mathbf{b}$ via Gaussian elimination, and still retain (some) stability?

(1) Randomly mix the columns by right multiplying $\mathbf{A}$ by a random unitary matrix $\mathbf{V}$:

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

(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.

Randomized algorithms for computing full 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 $\mathbf{G}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{G}$.
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.

Randomized algorithms for computing full factorizations of matrices

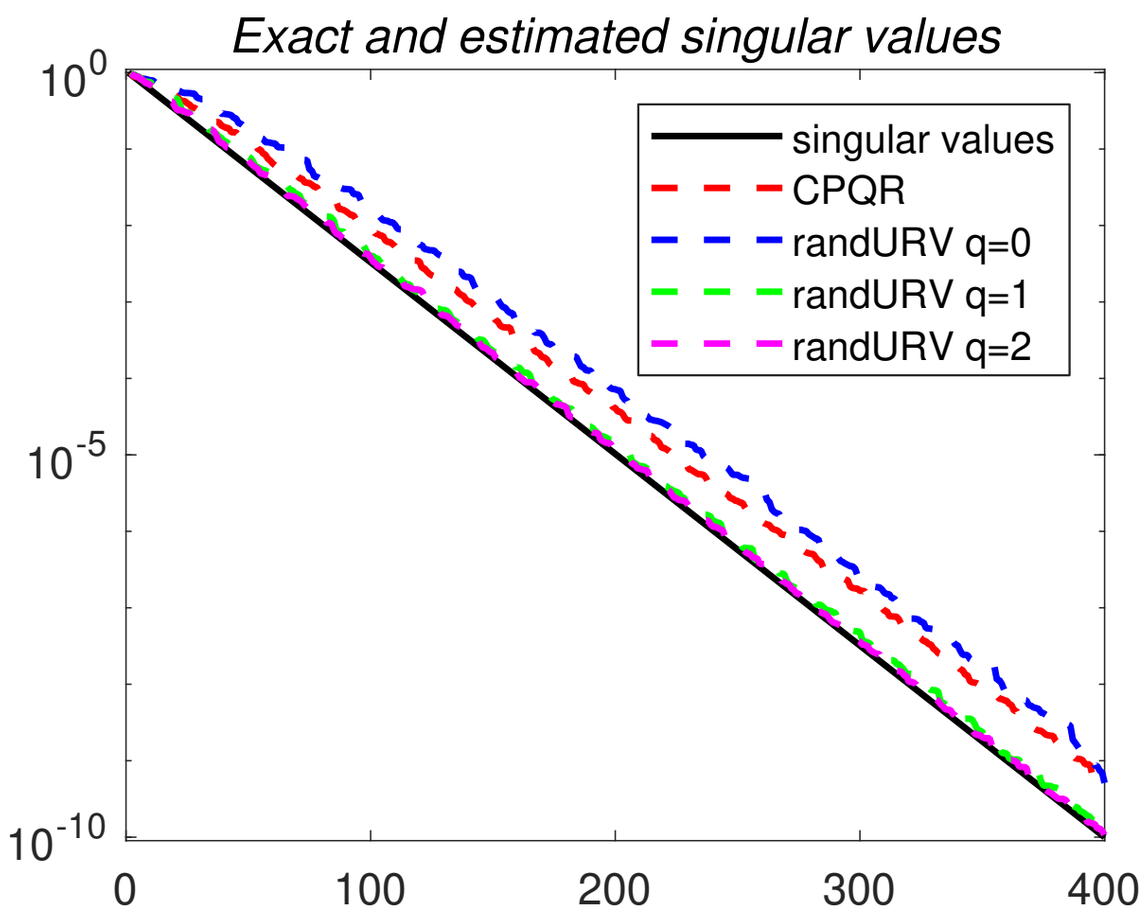
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

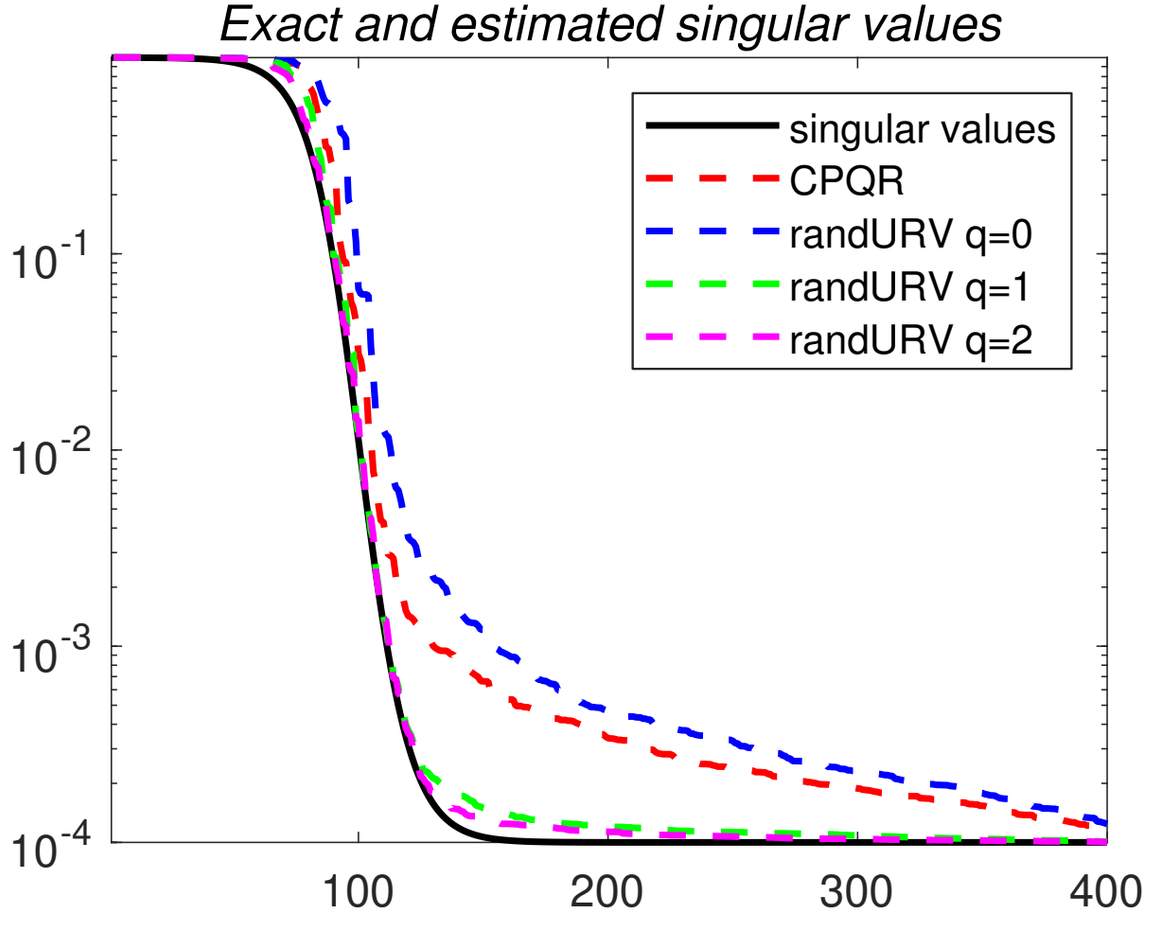
1. Draw a Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{G}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{V}\mathbf{R}_{\text{trash}}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

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.





Randomized algorithms for computing full factorizations of matrices

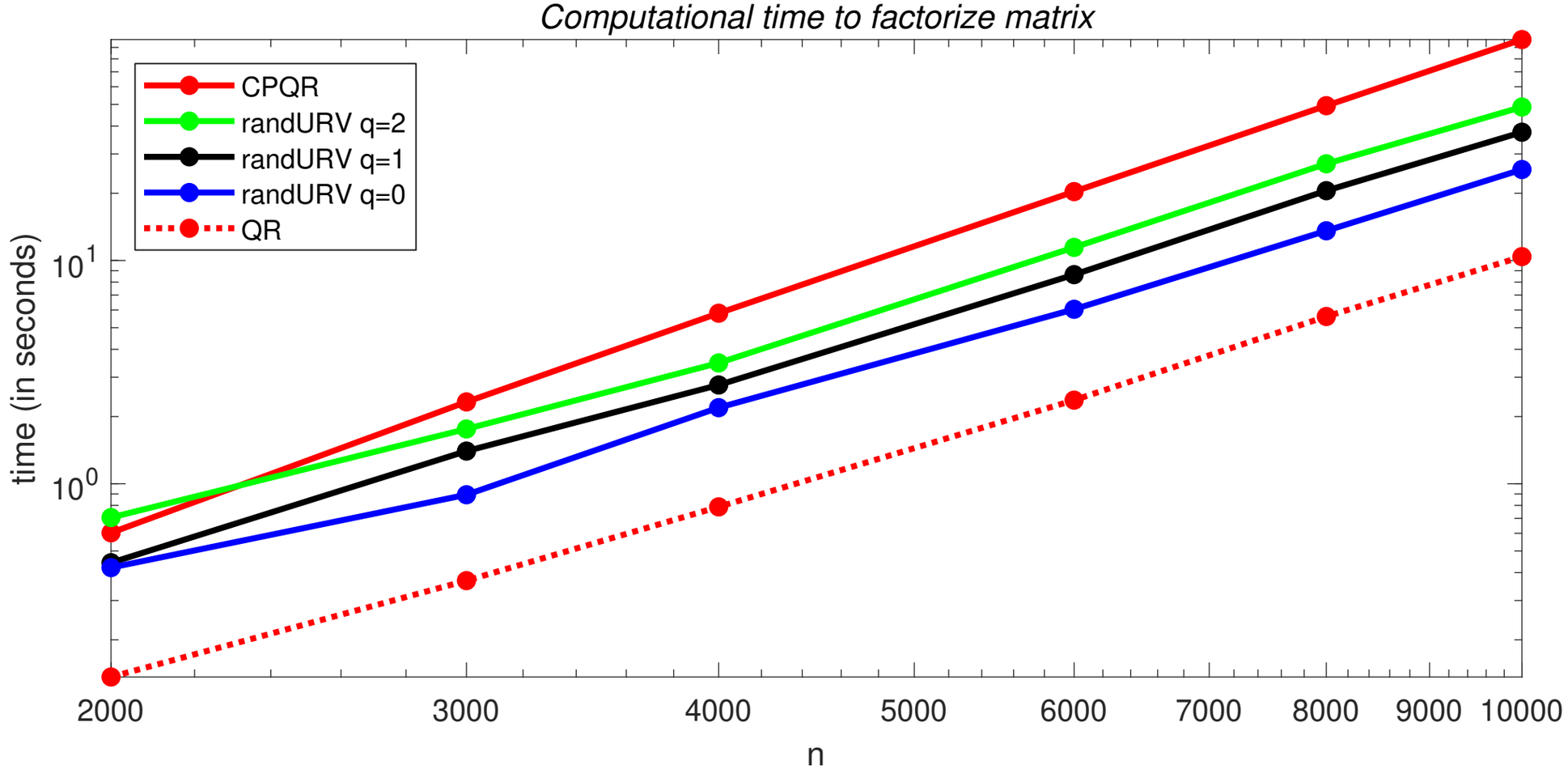
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw a Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{G}$.
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.



Randomized algorithms for computing full factorizations of matrices

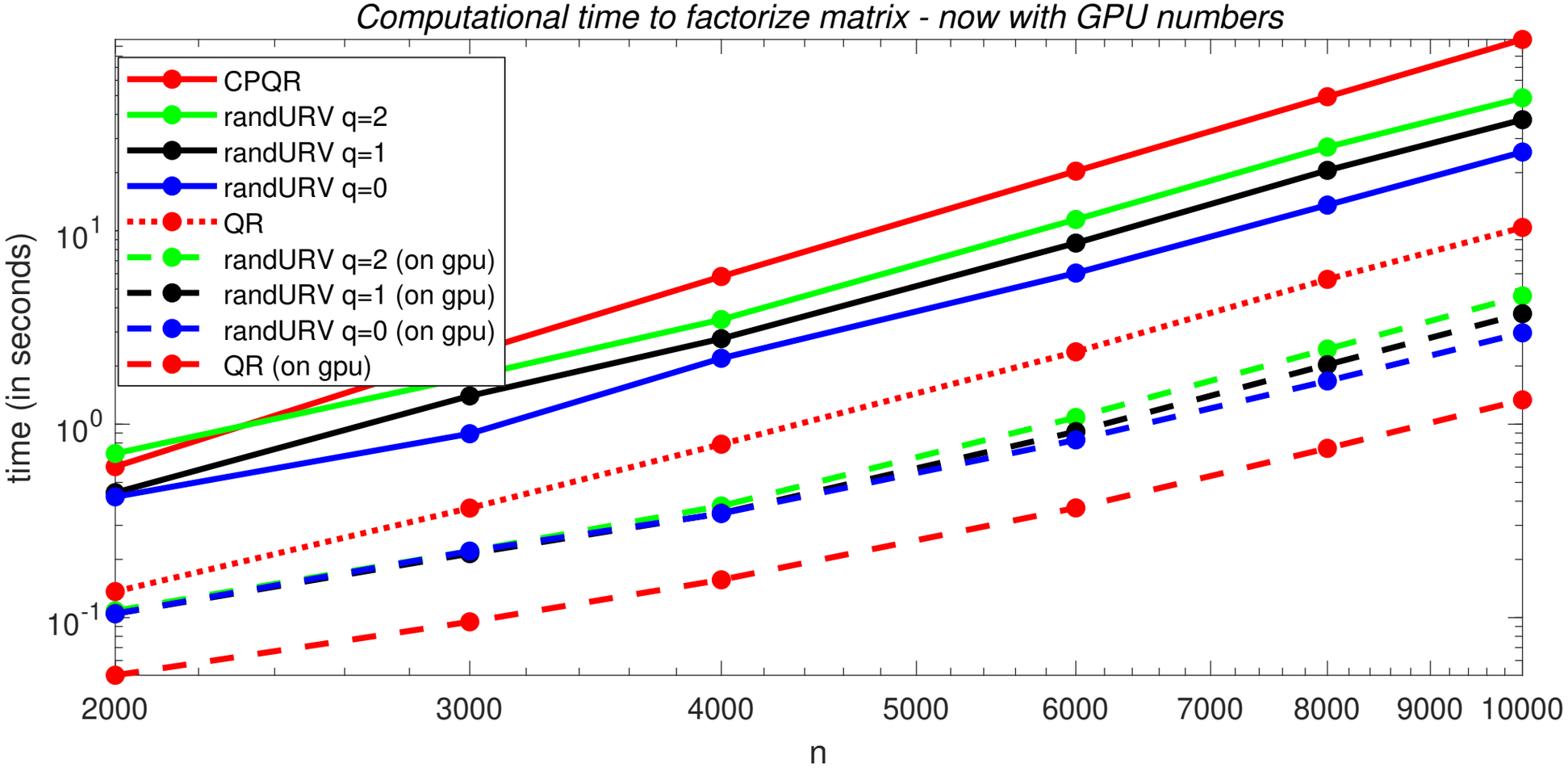
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw a Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{G}$.
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.



Randomized algorithms for computing full 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 $\mathbf{G}$ and form $\mathbf{Y} = \big(\mathbf{A}^*\mathbf{A}\big)^q\mathbf{G}$.
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);
```

See arxiv 1812.06007 — joint with with Abinand Gopal.

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.

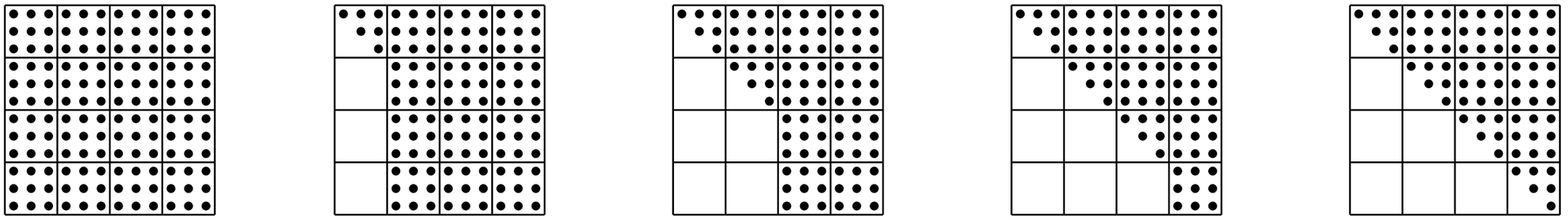
Randomized pivoting in Householder QR

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.

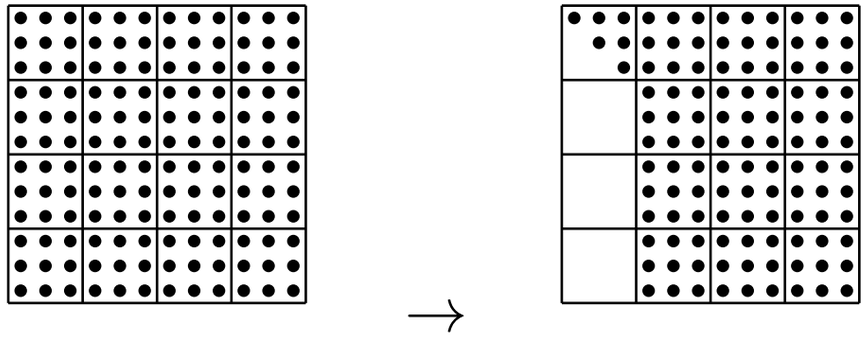
We seek $\mathbf{P}_j$ so that the set of b chosen columns *has maximal spanning volume.*

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

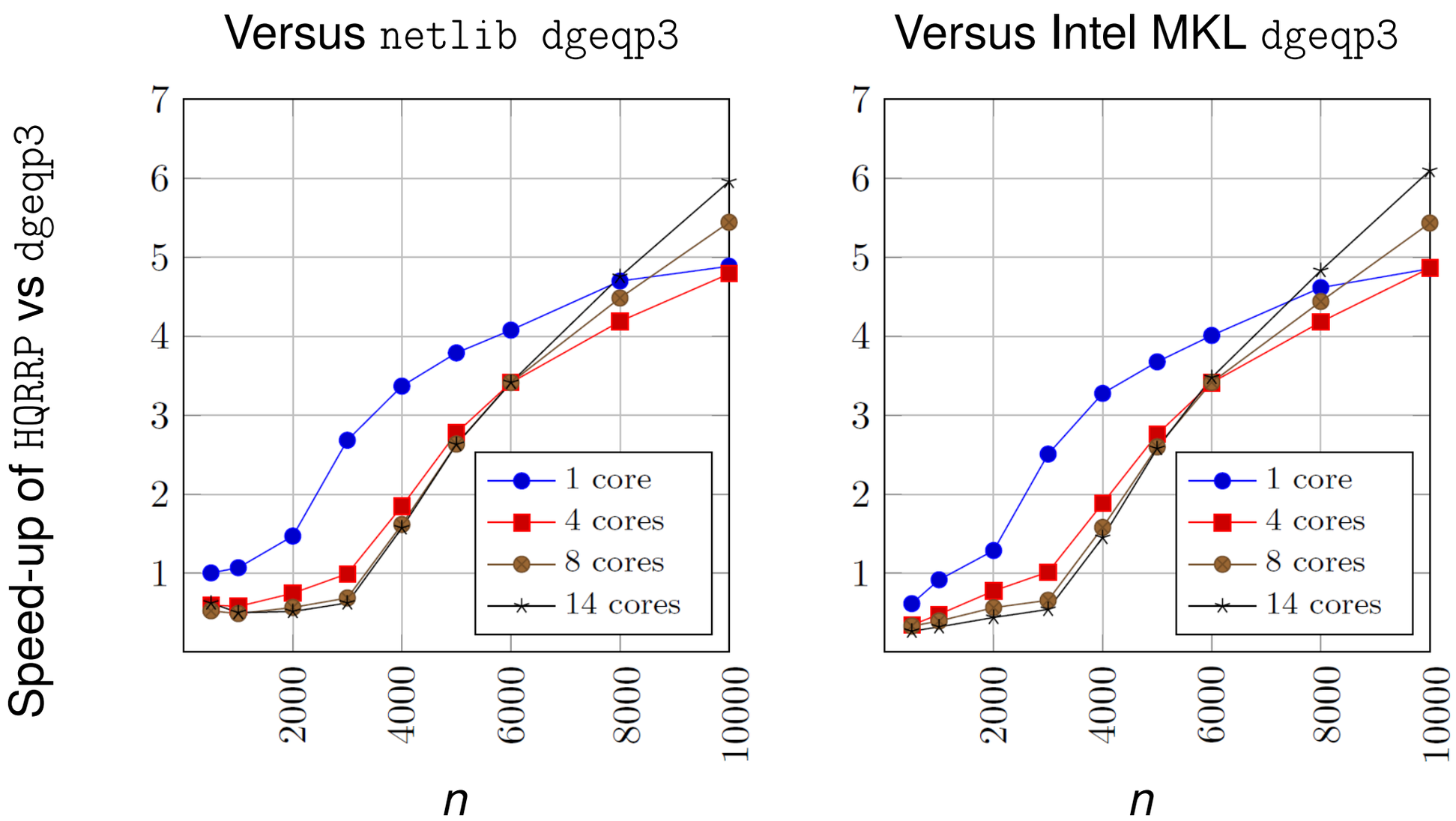
$$\underset{b \times n}{\mathbf{F}} = \underset{b \times m}{\mathbf{G}} \; \underset{m \times n}{\mathbf{A}}$$

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.

Towards SVD like optimality: The randUTV algorithm

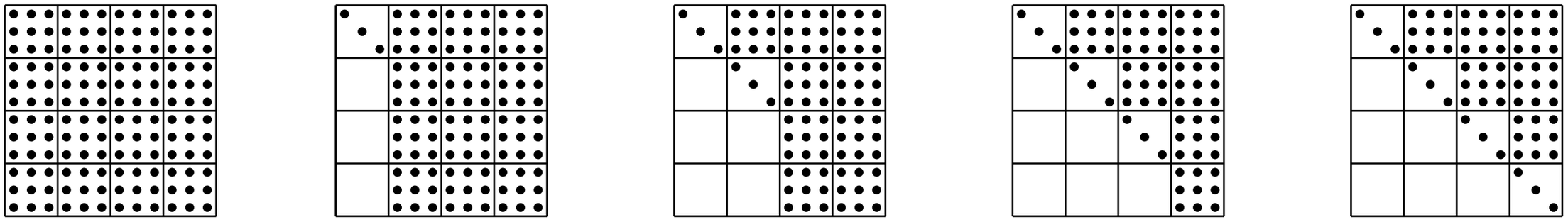
Given a dense $n \times n$ matrix $\mathbf{A}$, compute a factorization

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

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.

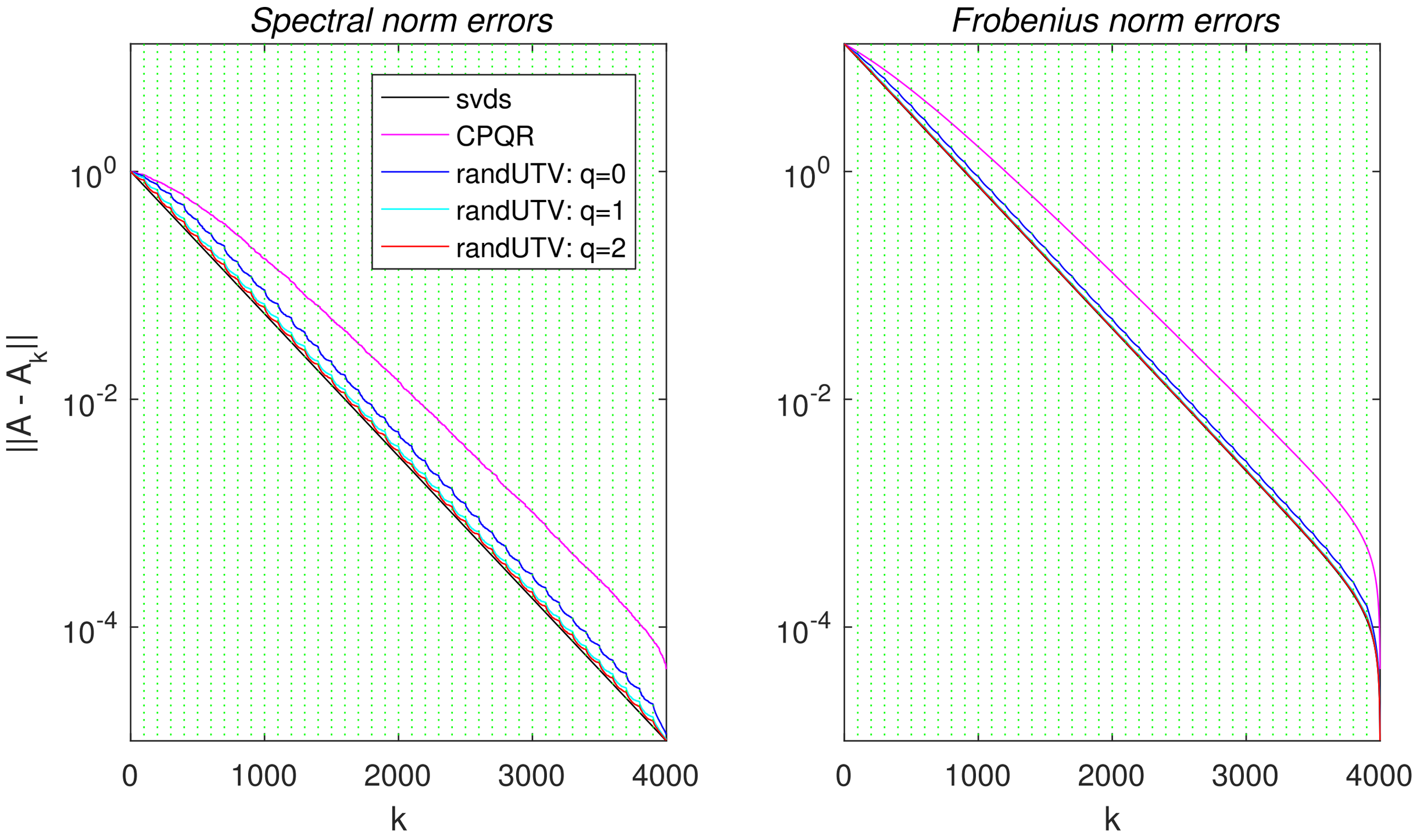
The technique proposed is based on a blocked version of classical Householder QR:



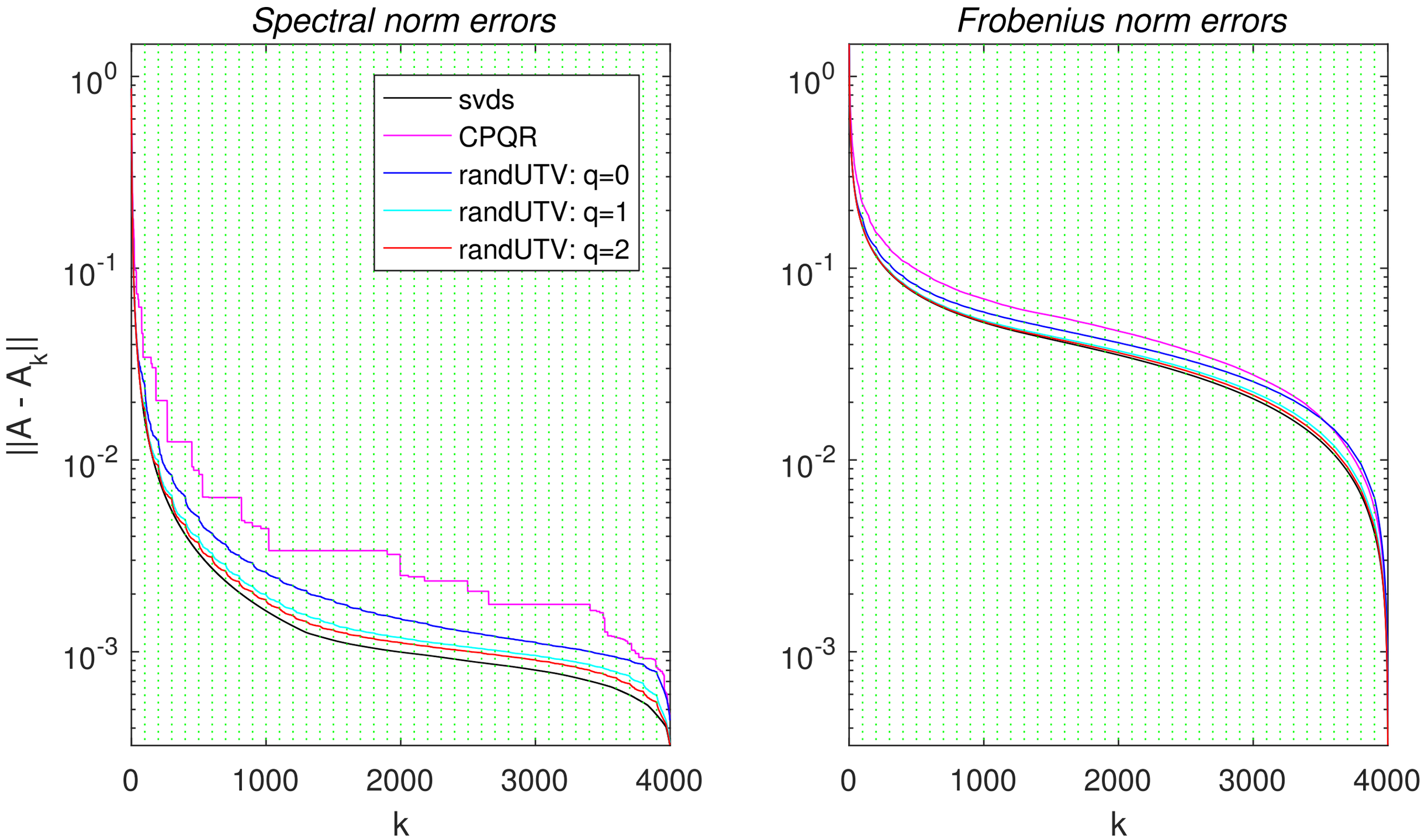
$$\mathbf{A}_0 = \mathbf{A} \qquad \mathbf{A}_1 = \mathbf{U}_1^* \mathbf{A}_0 \mathbf{V}_1 \qquad \mathbf{A}_2 = \mathbf{U}_2^* \mathbf{A}_1 \mathbf{V}_2 \qquad \mathbf{A}_3 = \mathbf{U}_3^* \mathbf{A}_2 \mathbf{V}_3 \qquad \mathbf{A}_4 = \mathbf{U}_4^* \mathbf{A}_3 \mathbf{V}_4$$

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

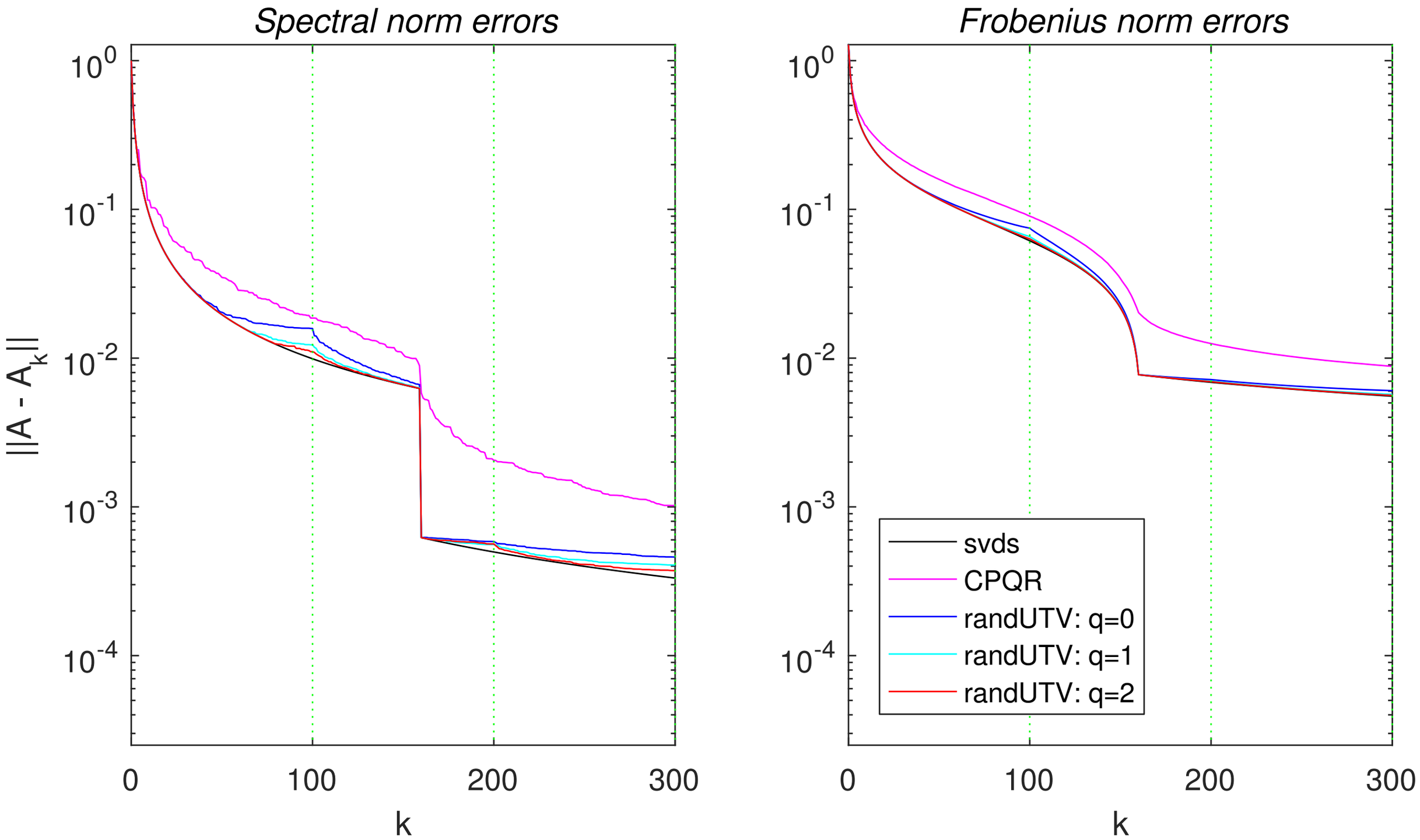
Our objective is in each step to find an approximation *to the linear subspace* spanned by the b dominant singular vectors of a matrix. The randomized range finder is perfect for this, especially when a small number of power iterations are performed. Easier and more natural than choosing pivoting vectors.



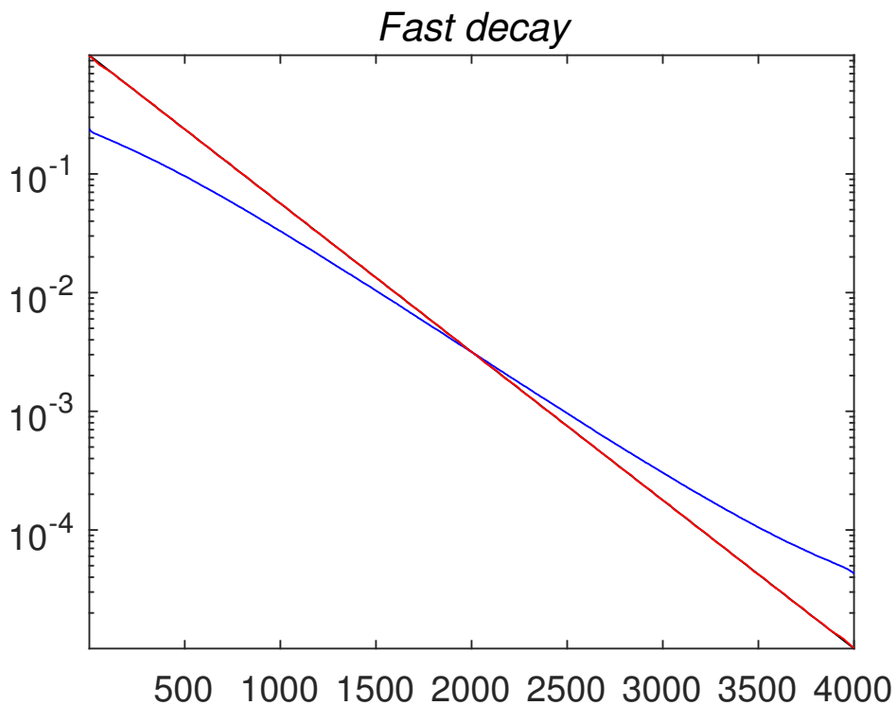
Rank-k approximation errors for the matrix "Fast Decay" of size 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*

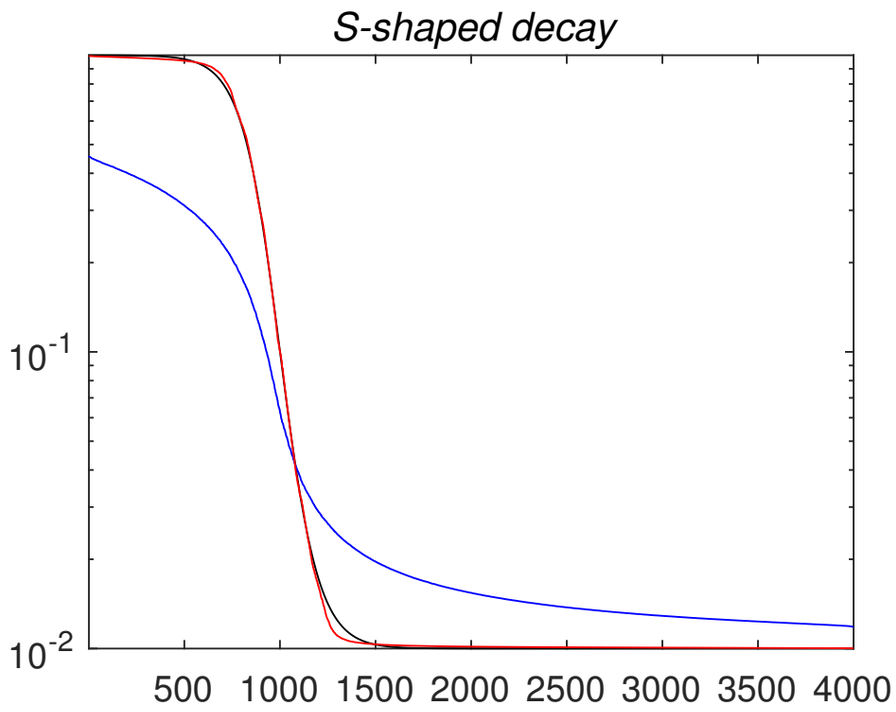


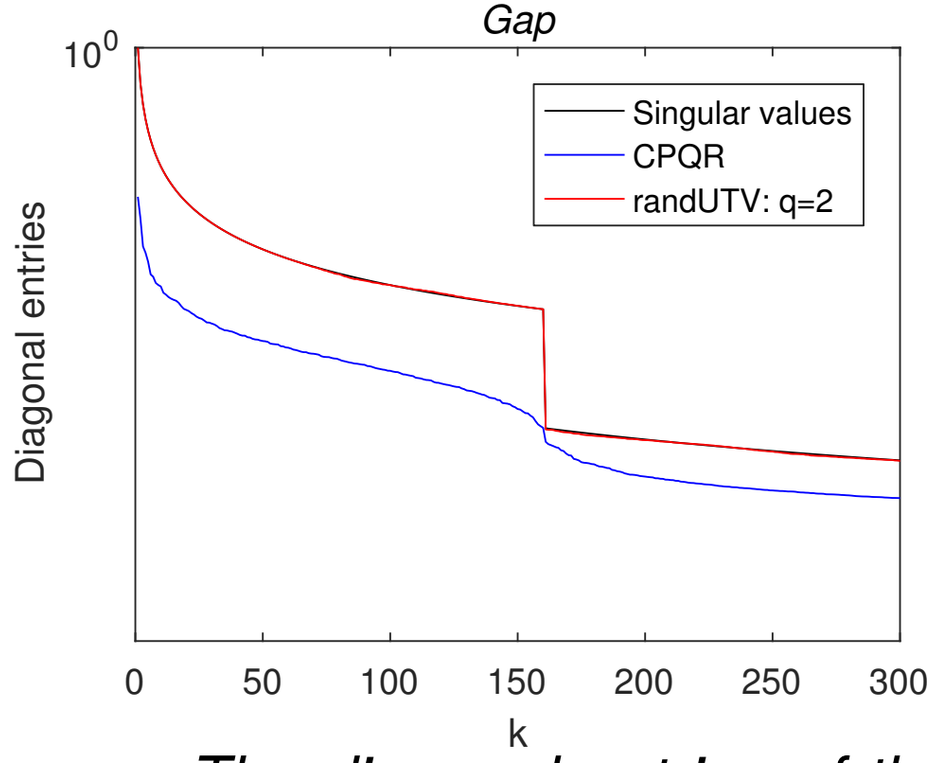
Rank-k approximation errors for the matrix "BIE" of size 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*

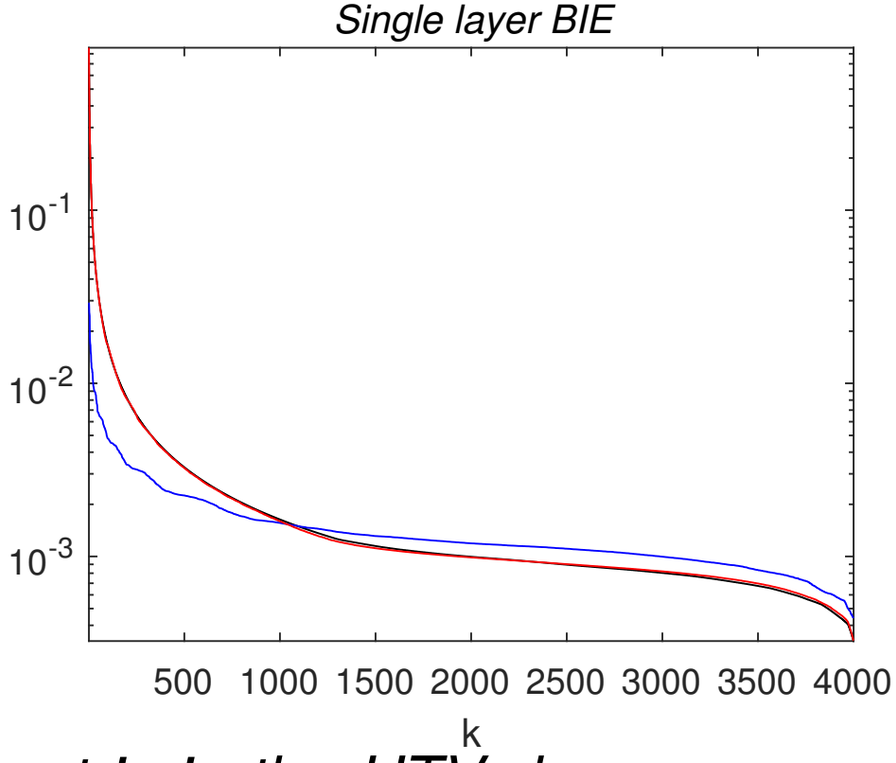


Rank-k approximation errors for $k \leq 300$ *for the matrix "Gap" of size* 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*





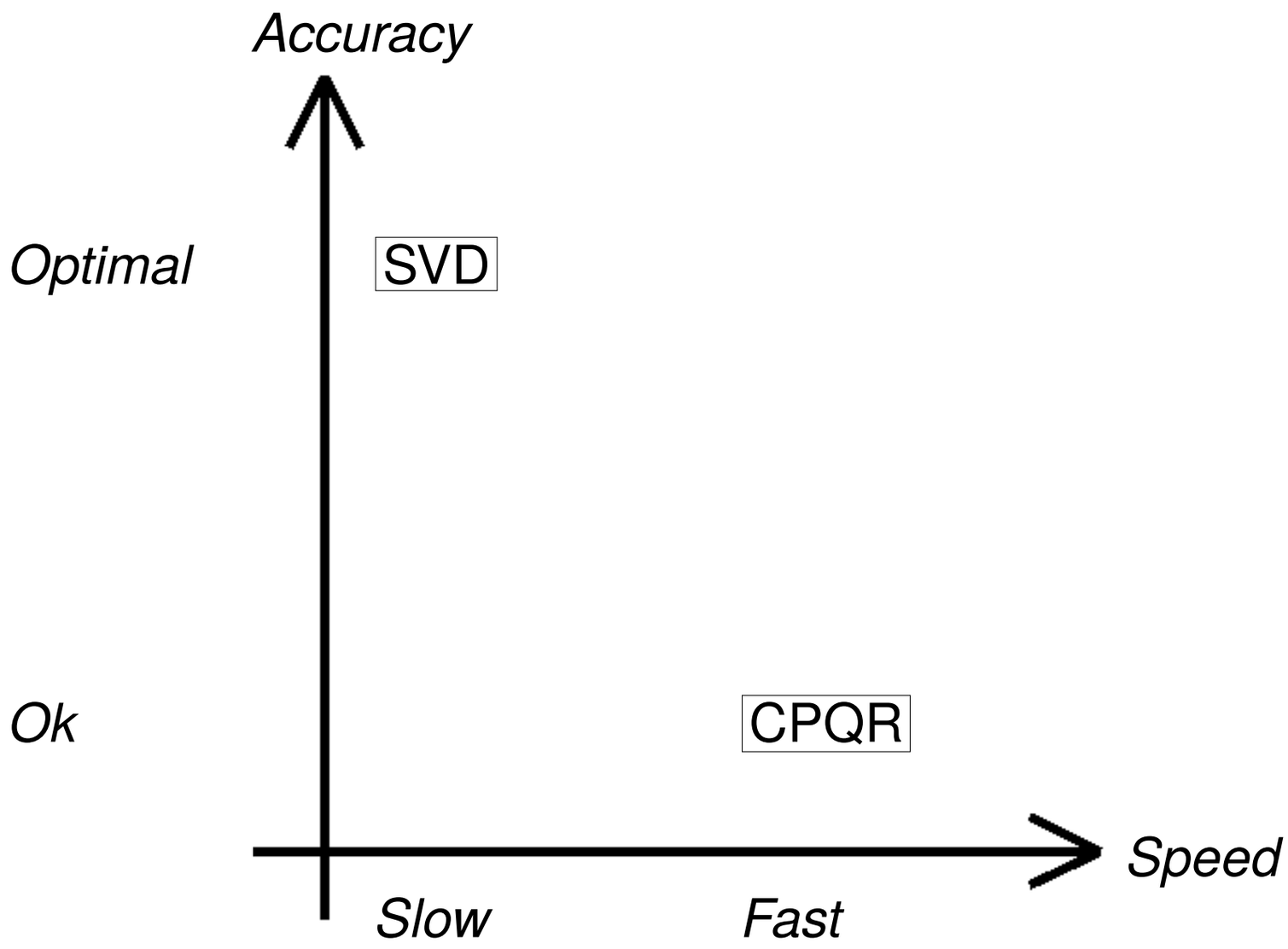




The diagonal entries of the **T***-matrix in the UTV decomposition (red) provide excellent approximations to the true singular values (black).*

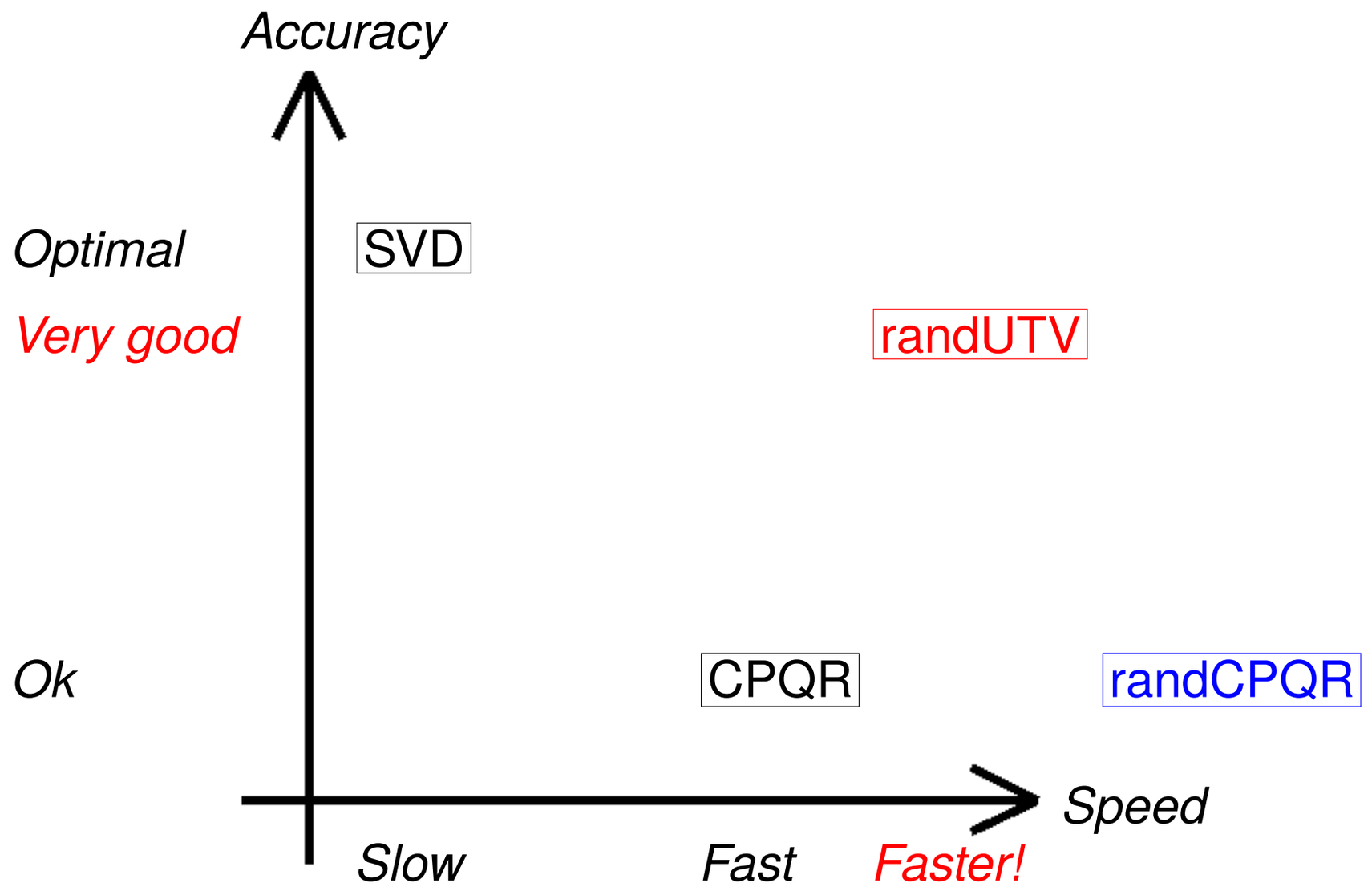
A comparison of algorithms for computing rank-revealing factorizations

For the task of computing rank revealing factorizations, the classical choice has been between SVD and column pivoted QR (CPQR). SVD is slow, and CPQR is inaccurate:



A comparison of algorithms for computing rank-revealing factorizations

For the task of computing rank revealing factorizations, the classical choice has been 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).

References on randomized algorithms for full factorizations:

Column pivoted QR:

- Martinsson arXiv:1505.08115, 2015.
- Duersch/Gu arXiv:1509.06820, 2015.
- Martinsson/Quintana-Ortí/Heavner/van de Geijn SISC 2017.
- Xiao/Gu/Langou, IEEE HiPC 2017. (And arXiv:1804.05138.)
- Duersch/Gu SISC 2017 and SIREV 2020.

UTV:

- Martinsson/Quintana-Ortí/Heavner, ACM TOMS, 2019.
- Heavner/Martinsson/Quintana-Ortí arXiv:2002.06960, 2020.

Fully pivoted LU:

- Melgaard/Gu arXiv:1511.08528, 2015.

Survey:

- Martinsson/Tropp, Acta Numerica 2020. (And arxiv:2002.01387)

Beating $O(n^3)$: 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.

Key points:

- Interpolatory and CUR decompositions are useful and popular.
  - Preserve properties like sparsity and non-negativity.
  - Good for data interpretation.
  - Storage efficient.
  - Invaluable in the context of modern Fast Multipole Methods and Fast Direct Solvers.

- Randomized sketching is an excellent tool for computing the CUR/ID.
  - Particularly effective for large sparse matrices.
  - Great for huge matrices stored out of core.
  - Improved asymptotic flop count — “fast Johnson-Lindenstrauss transforms”.
  - As robust and accurate as deterministic methods.

  Keep it simple, however! Gaussian sketch + partially pivoted LU excel together.

- Randomized algorithms for accelerating full factorizations of matrices.
  - Enable *blocking* of column pivoted QR.
  - Order of magnitude acceleration in some environments.
  - The “randomized UTV” algorithm is very fast, and almost as precise as the SVD.
    Particularly effective for out-of-core, GPU, distributed memory, etc.

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

Surveys:

- 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 rather than theory.

- 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`