Randomized projection methods for reducing communication in matrix computations

Per-Gunnar Martinsson

Department of Mathematics

University of Texas at Austin

Students & postdocs: Tracy Babb, Abinand Gopal, Nathan Halko, Nathan Heavner, Sergey Voronin, Anna Yesypenko, Patrick Young.

Collaborators: Robert van de Geijn, Gregorio Quintana-Ortí.

Research support by:

Randomized dimension reduction

Let $\{\mathbf{a}^{(j)}\}_{j=1}^{n}$ be a set of points in $\mathbb{R}^m$, where m is very large. Consider tasks such as:

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

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

Idea: Find a "nice" embedding $f : \mathbb{R}^m \to \mathbb{R}^d$ for $d \ll m$ that is almost isometric:

$$\|f(\mathbf{a}^{(i)}) - f(\mathbf{a}^{(j)})\| \approx \|\mathbf{a}^{(i)} - \mathbf{a}^{(j)}\|, \qquad \forall i, j \in \{1, 2, 3, \ldots, n\}.$$

Then solve the problems for the vectors $\{f(\mathbf{a}^{(j)})\}_{j=1}^{n}$ in $\mathbb{R}^d$.

Lemma [Johnson-Lindenstrauss]: For $d \sim \log(n)$, there exists an orthogonal projection that "approximately" preserves distances.

To be precise, we have:

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

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

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

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

You can prove that if you pick d as specified, and repeatedly draw a Gaussian random matrix $\mathbf{G}$ of size $d \times m$, then the likelihood is larger than zero that the map

$$f(\mathbf{x}) = \frac{1}{\sqrt{d}} \mathbf{G}\, \mathbf{x}$$

satisfies the criteria.

Practical problem: You have two bad choices:

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

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

Question: Is it possible to build algorithms that combine the powerful dimension reduction capability of randomized projections with the accuracy and robustness of classical deterministic methods?

Question: Is it possible to build algorithms that combine the powerful dimension reduction capability of randomized projections with the accuracy and robustness of classical deterministic methods?

Putative answer: Yes — use a two-stage approach:

(A) *Randomized pre-conditioner:*

In a pre-computation, random projections are used to create low-dimensional sketches of the high-dimensional data. These sketches are somewhat distorted, but approximately preserve key properties to very high probability.

(B) *Deterministic post-processing:*

Once a sketch of the data has been constructed in Stage A, classical deterministic techniques are used to compute desired quantities to very high accuracy, *starting directly from the original high-dimensional data*.

Example 1 of two-stage approach: Nearest neighbor search in $\mathbb{R}^m$

Objective: Suppose you are given n points $\{\mathbf{a}^{(j)}\}_{j=1}^{n}$ in $\mathbb{R}^m$. The coordinate matrix is

$$\mathbf{A} = \left[\mathbf{a}^{(1)}\ \mathbf{a}^{(2)}\ \ldots\ \mathbf{a}^{(n)}\right] \in \mathbb{R}^{m\times n}.$$

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

If m is "small" (say $m \leq 10$ or so), then you have several options; you can, e.g, sort the points into a tree based on hierarchically partitioning space (a "kd-tree").

Problem: Classical techniques of this type get very expensive as m grows.

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

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

Example 1 of two-stage approach: Nearest neighbor search in $\mathbb{R}^m$

Objective: Suppose you are given n points $\{\mathbf{a}^{(j)}\}_{j=1}^{n}$ in $\mathbb{R}^m$. The coordinate matrix is

$$\mathbf{A} = \left[\mathbf{a}^{(1)}\ \mathbf{a}^{(2)}\ \ldots\ \mathbf{a}^{(n)}\right] \in \mathbb{R}^{m\times n}.$$

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

(A) *Randomized probing of data:* Use a Johnson-Lindenstrauss random projection to map the n-particle problem in $\mathbb{R}^m$ (where m is large) to an n-particle problem in $\mathbb{R}^d$ where $d \sim \log n$. Run a deterministic nearest-neighbor search in $\mathbb{R}^d$ and store a list of the ℓ nearest neighbors for each particle (for simplicity, one can set $\ell = k$). Then repeat the process several times. If for a given particle a previously undetected neighbor is discovered, then simply add it to a list of potential neighbors.

(B) *Deterministic post-processing:* The randomized probing will result in a list of putative neighbors that typically contains more than k elements. But it is now easy to compute the pairwise distances in the original space $\mathbb{R}^m$ to judge which candidates in the list are the k nearest neighbors.

Jones, Osipov, Rokhlin, 2011

Example 2 of two-stage approach: Randomized SVD

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

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

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal.

(A) *Randomized pre-conditioner:*

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.

Observe that 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. The risk of missing the dominant ones is for practical purposes zero.

Example 2 of two-stage approach: Randomized SVD

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

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

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{D}$ is diagonal.

Fix an over-sampling parameter p. Say $p = 10$.

(A) *Randomized pre-conditioner:*

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

A.2 Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{G}$. `Y = A * G`

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

(B) *Deterministic post-processing:*

B.1 Form the $(k+p) \times n$ 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, Sigma, V] = svd(B,'econ')`

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

(Truncate the last p terms in step B.2 to attain a factorization of precise rank k.)

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 $\mathbf{G}$.

(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.

(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.

(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.

(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

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

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

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

- Analogous schemes exist for computing a partial QR factorization, or a so called "interpolative decomposition" where a number of the columns/rows are chosen to serve as a basis for the column/row space.

  $\rightarrow$ *Relaxed solution to "maximal spanning volume" problem on first slide.*

- 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 $\mathbf{Y} = \mathbf{A}\big(\mathbf{A}^*\big(\mathbf{AG}\big)\big)$ instead of $\mathbf{Y} = \mathbf{AG}$.

- We can reduce the flop count from $O(mnk)$ to $O(mn\log k)$ by using a so called "fast Johnson-Lindenstrauss" transform. Practical speed gain too!

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 $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(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 $\mathbf{G}$. We have rigorous mathematical results describing the errors of the algorithm in expectation, as well as the risk of large deviations. Connections to random matrix theory.

The perhaps most important feature of randomized algorithms is that they are very communication efficient. This makes them particularly competitive in strongly communication constrained environments (huge matrices stored out-of-core, distributed memory parallel computers, GPUs).

There exist *single-pass* versions of the RSVD that work even under the constraint that each matrix element can be viewed only once. ("Streaming algorithms.")

Recent result: Randomization can be used to greatly accelerate *full* rank-revealing factorizations such as the column pivoted QR factorization, or the UTV factorization. The gain is attained due to decreased communication, not fewer flops.

Accelerate algorithms for FULL factorizations of matrices

Starting point (Demmel, Dumitriu, Holtz, 2007): Let $\mathbf{A}$ be an $n \times n$ matrix. We seek a rank-revealing UTV factorization $\mathbf{A} = \mathbf{UTV}^*$, with $\mathbf{U}, \mathbf{V}$ unitary, and $\mathbf{T}$ triangular.

Proceed as follows:

- Draw an $n \times n$ Gaussian matrix $\mathbf{G}$ and orthonormalize its columns $[\mathbf{V}, \sim] = \mathtt{qr}(\mathbf{G})$.
- Form a QR factorization of $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UT}$.

Then $\mathbf{A} = \mathbf{UTV}^*$ is provably "rank-revealing." But in a *very* weak sense.

Improved Demmel UTV (with power iteration): Same set-up.

- Draw an $n \times n$ Gaussian matrix $\mathbf{G}$ and compute $\mathbf{Y} = \left(\mathbf{A}^*\mathbf{A}\right)^q \mathbf{G}$ for $q = 1$ or 2.
- Orthonormalize the columns of $\mathbf{Y}$ so that $[\mathbf{V}, \sim] = \mathtt{qr}(\mathbf{Y})$.
- Form a QR factorization of $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UT}$.

Then $\mathbf{A} = \mathbf{UTV}^*$ is "rank-revealing." Very good for $q = 1$. Excellent for $q = 2$.

These algorithms require a huge number of flops!
But much faster in practice than, say, CPQR.

Key fact: The matrix-matrix multiply can be done *very* rapidly in many environments . . . GPU, distributed memory, fast algorithms, Strassen, etc.

Numerical results for the "Demmel URV" factorization

There are many different ways to measure the quality of a rank-revealing factorization. Let us describe one common measure: Let $\mathbf{A}$ be an $n \times n$ matrix factored as

$$\mathbf{A} = \mathbf{UTV}^*$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary, and where $\mathbf{T}$ is upper triangular. Define for $k \in \{1, 2, \ldots, n-1\}$ the quantities

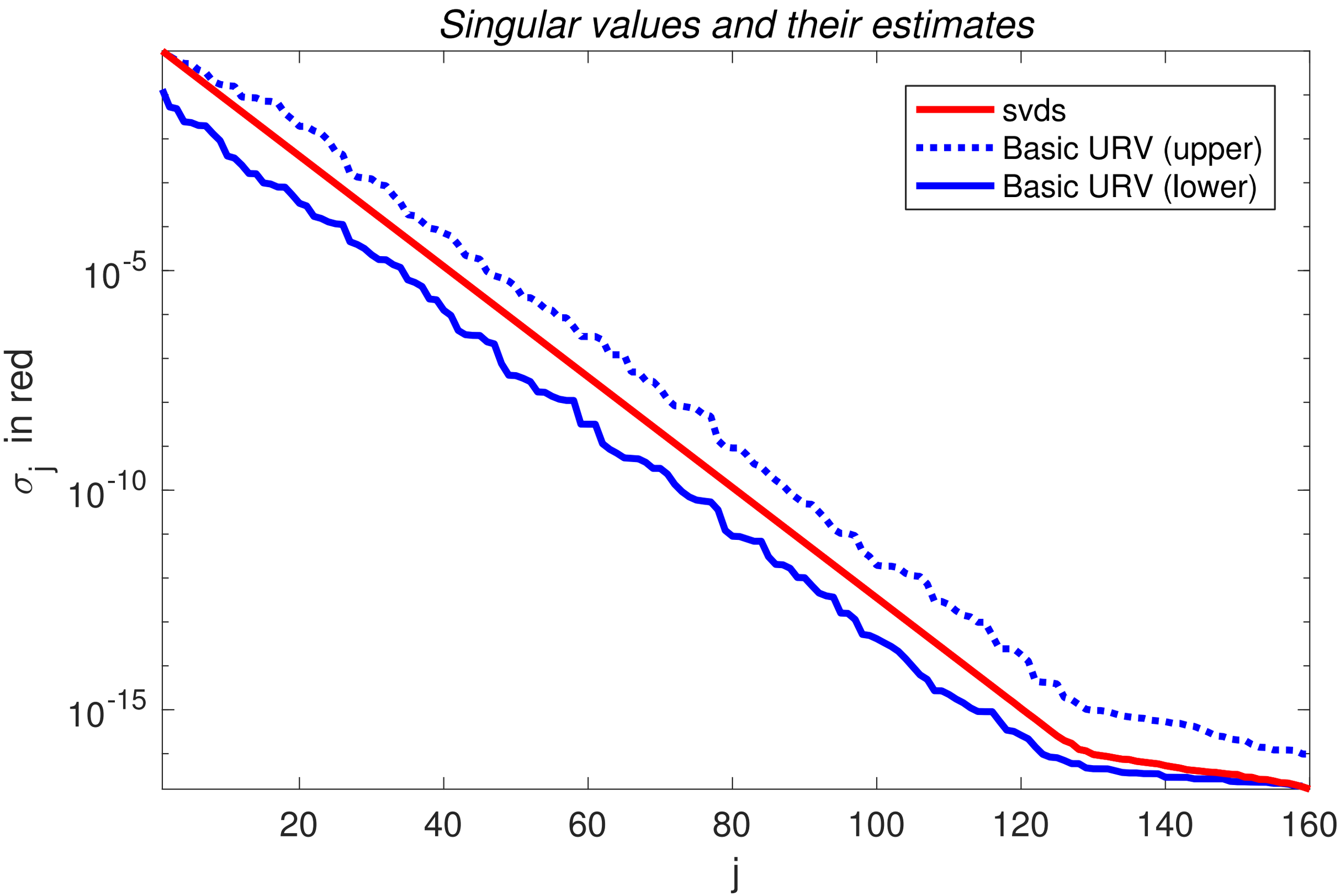
$$\nu_k = \sigma_k(\mathbf{T}(1:k, 1:k)),$$
$$\tau_{k+1} = \sigma_1(\mathbf{T}((k+1):n, (k+1):n)),$$

where $\sigma_j(\mathbf{X})$ denotes the j'th singular value of $\mathbf{X}$. One can easily prove that

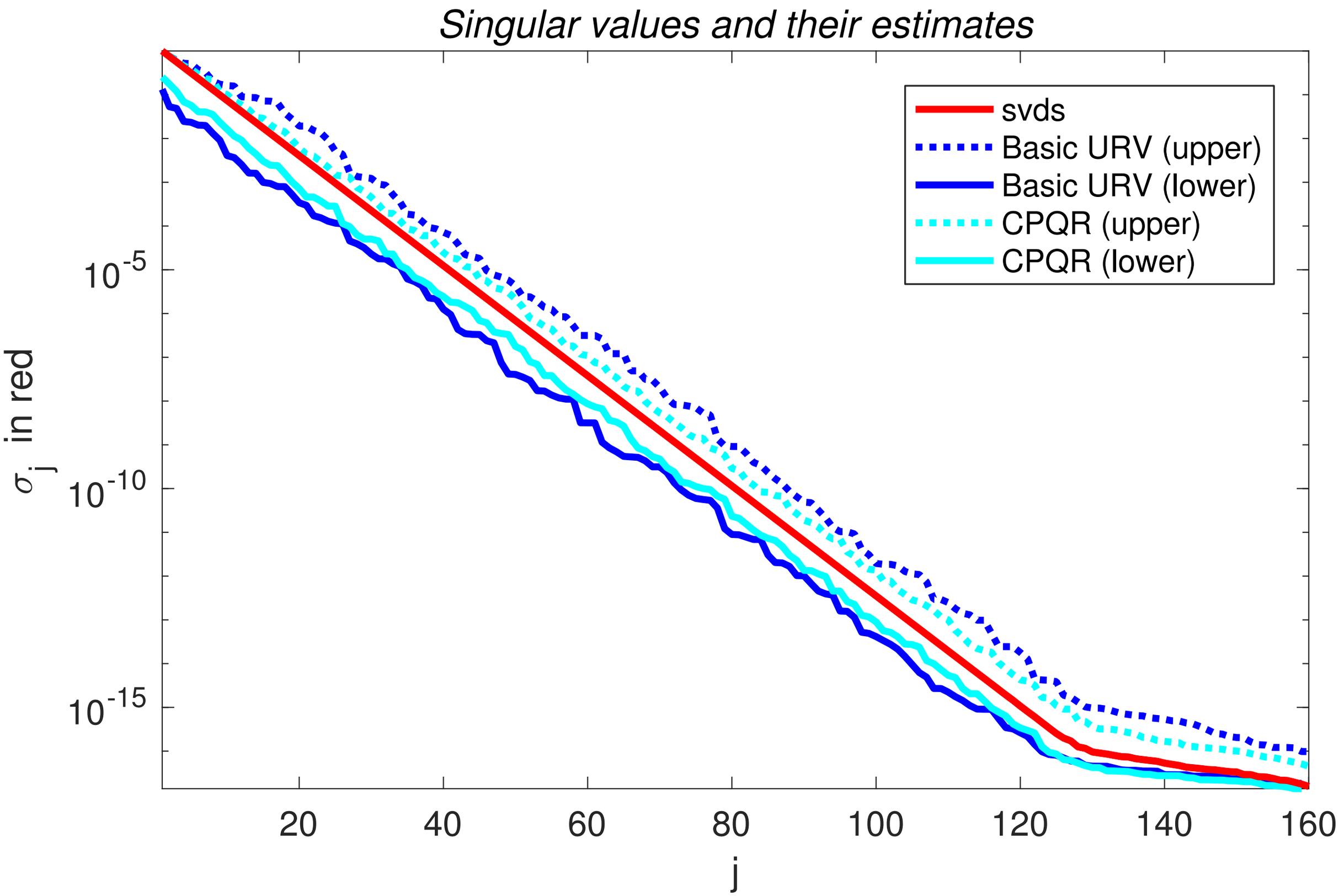
$$\nu_k \leq \sigma_k(\mathbf{A}) \leq \tau_k.$$

The more tightly that (ν_k, τ_k) constrains the k'th singular value, the better.

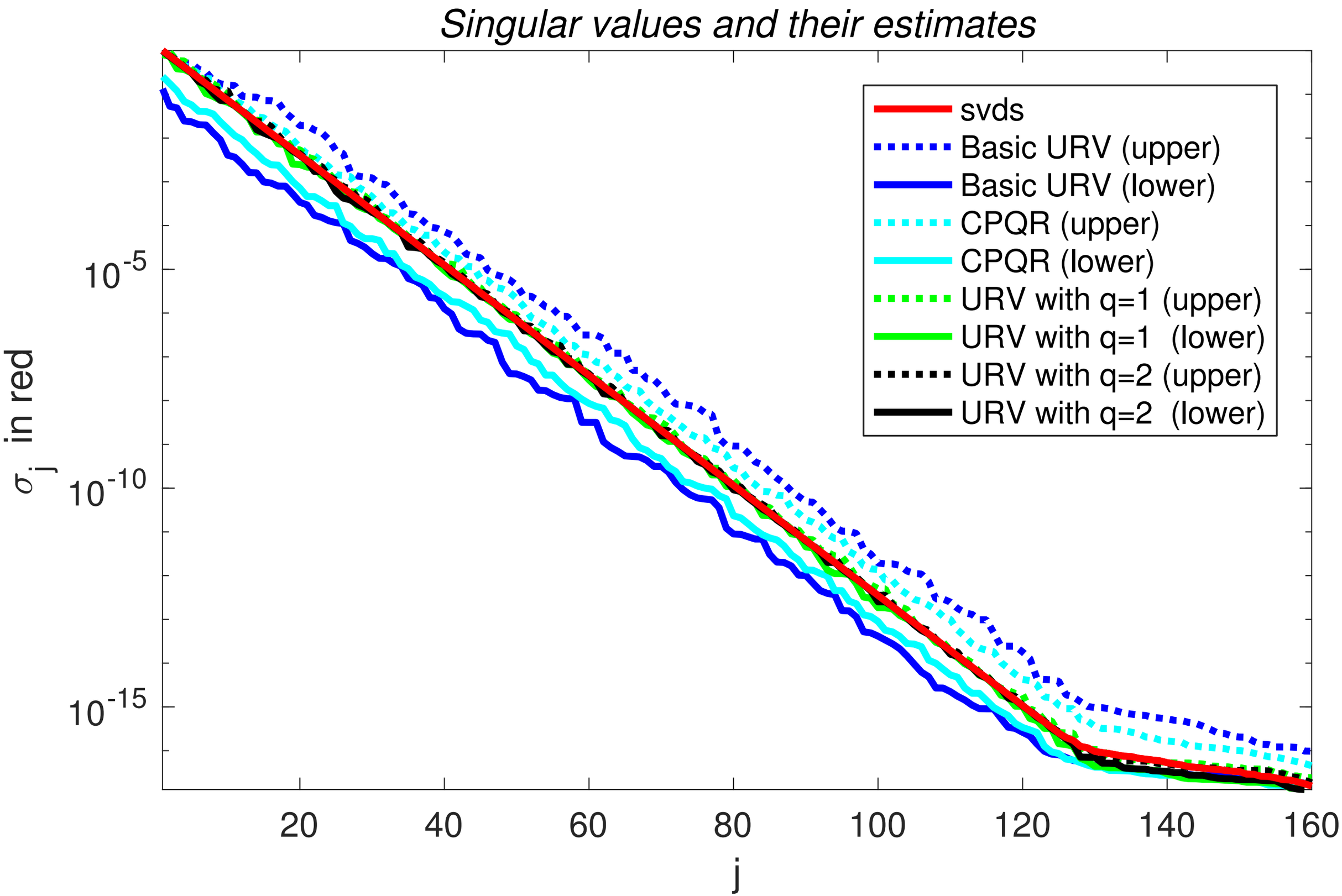
Numerical experiments illustrating the errors in the URV factorization



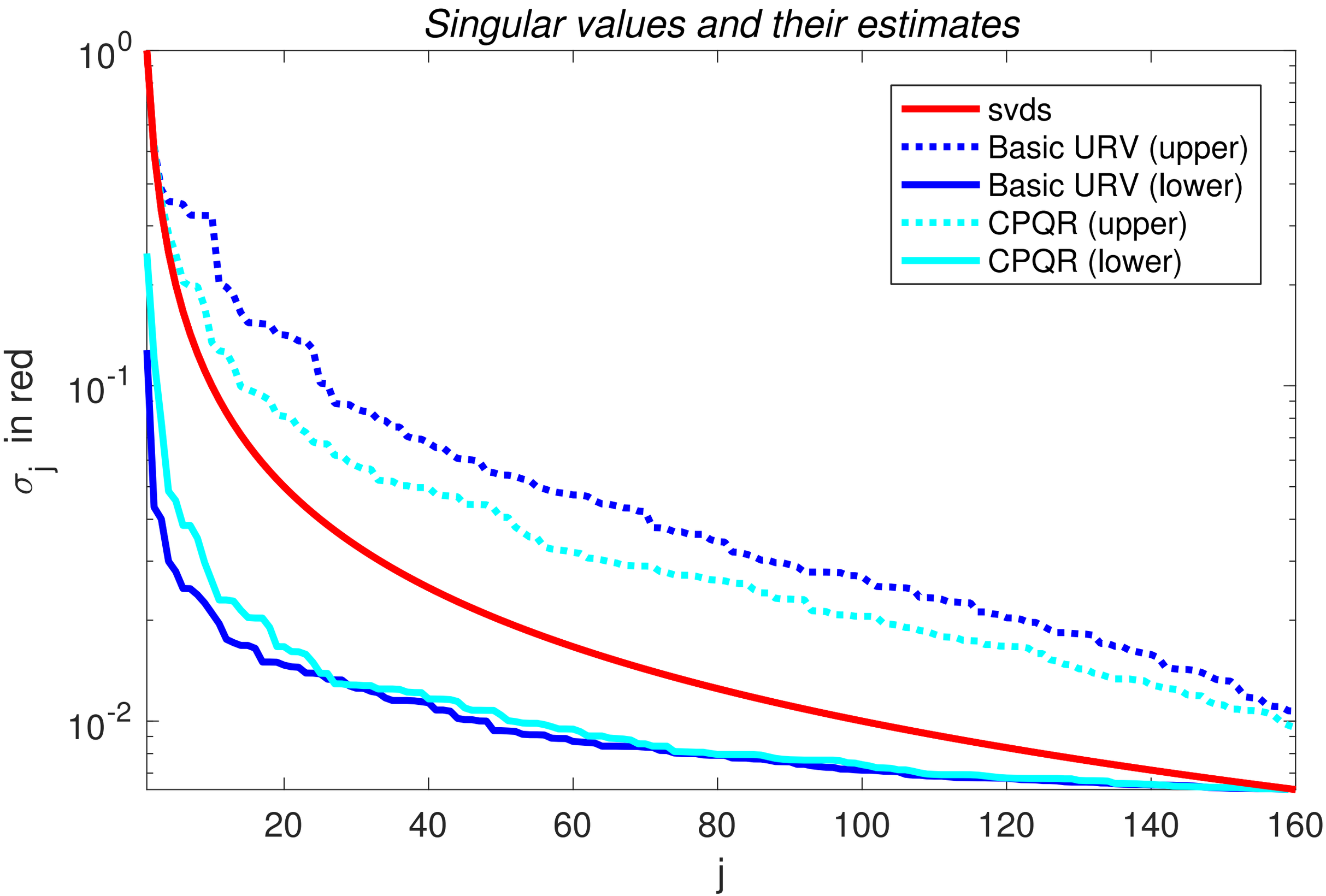
Numerical experiments illustrating the errors in the URV factorization



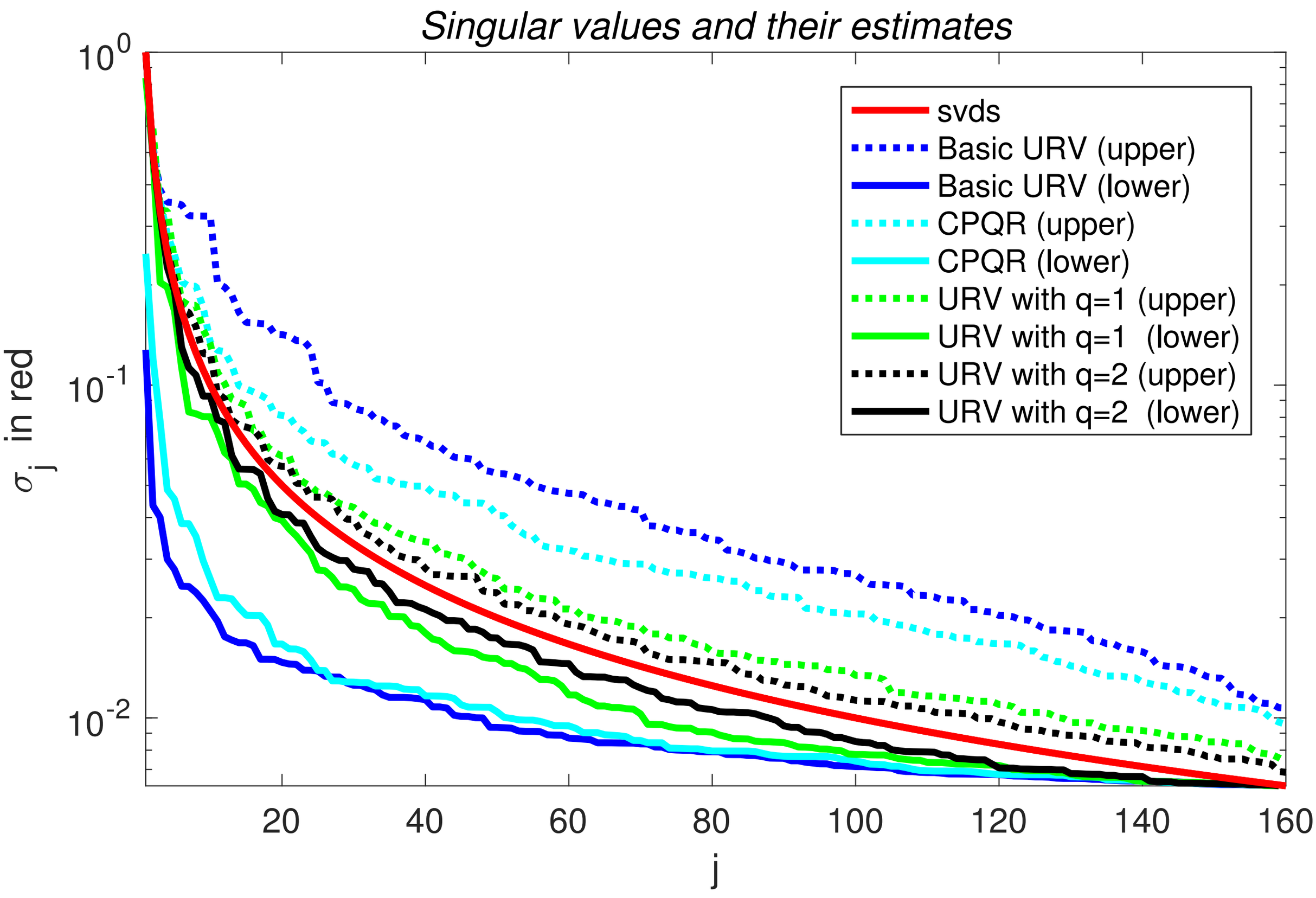
Numerical experiments illustrating the errors in the URV factorization



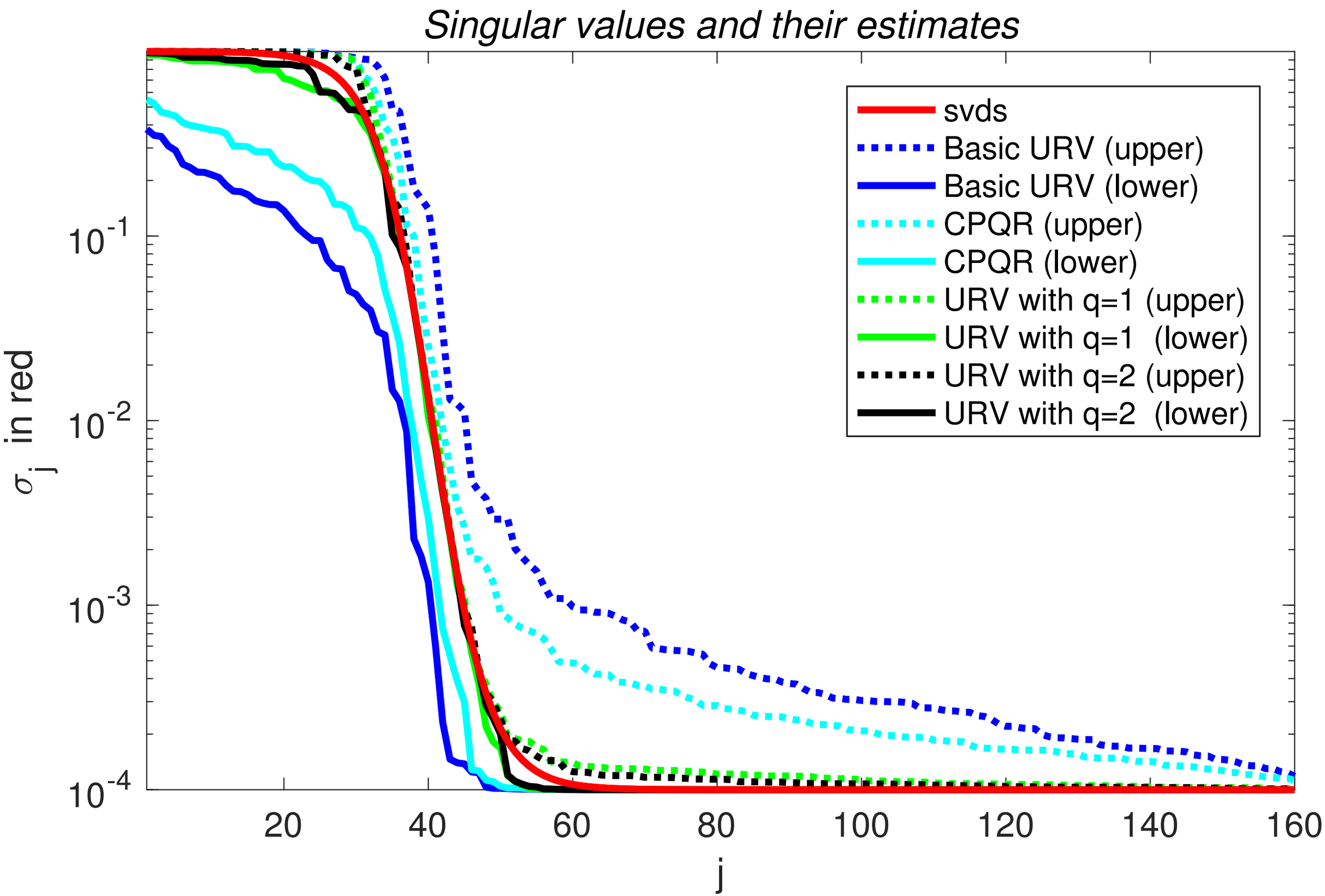
Numerical experiments illustrating the errors in the URV factorization



Numerical experiments illustrating the errors in the URV factorization



Numerical experiments illustrating the errors in the URV factorization



The UTV decomposition: A rank-revealing factorization

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

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array} \tag{3}$$

where $\mathbf{T}$ is upper triangular, and $\mathbf{U}$ and $\mathbf{V}$ are unitary. We want a factorization that is "rank-revealing", in the sense its truncation to a rank-k approximation should be of close to optimal accuracy. We also would like for the diagonal entries of $\mathbf{T}$ to approximate the singular values of $\mathbf{A}$, and for the off-diagonal entries to be very small.

A rank-revealing factorization has many uses:

- Finding a low-rank approximation to a matrix. (Obviously!)
- Solving ill-conditioned linear systems, or linear regression problems.
- Finding bases for fundamental subspaces.

Basically, when (3) is rank-revealing, it can be used for almost anything that the SVD is recommended for.

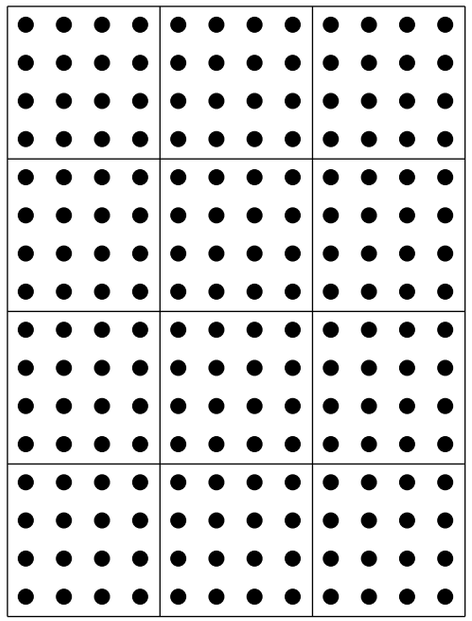
The UTV decomposition: Overview of proposed algorithm randUTV

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

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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

The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



$$\mathbf{A}_0 = \mathbf{A}$$

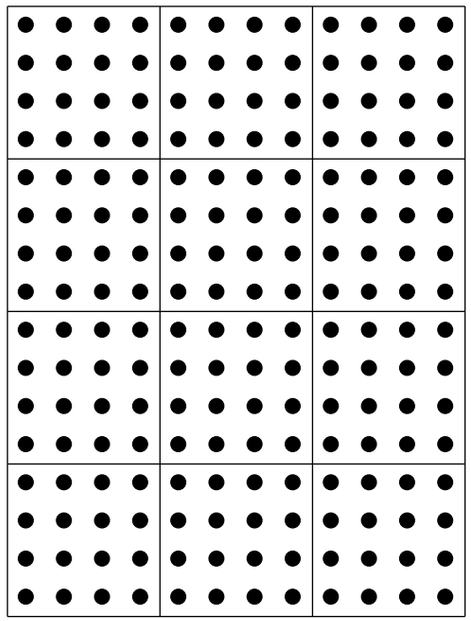
The UTV decomposition: Overview of proposed algorithm randUTV

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

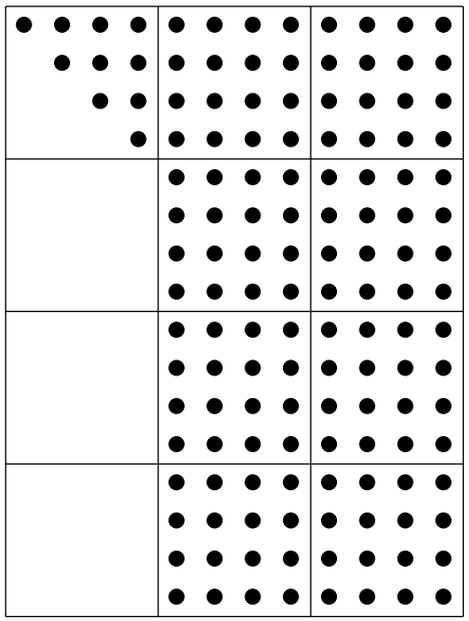
$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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

The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



$\mathbf{A}_0 = \mathbf{A}$



$\mathbf{A}_1 = \mathbf{U}_1^* \mathbf{A}_0 \mathbf{V}_1$

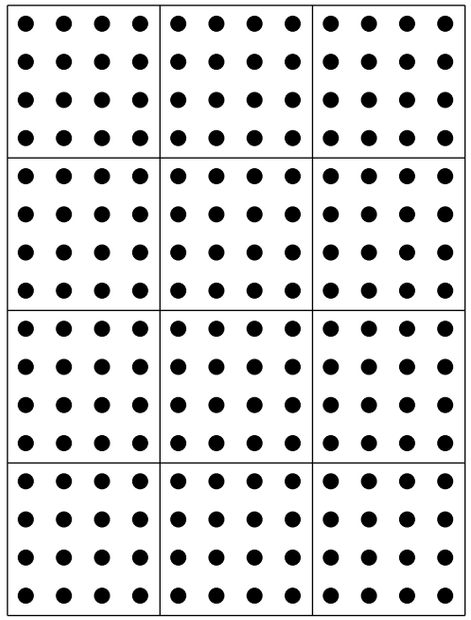
The UTV decomposition: Overview of proposed algorithm randUTV

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

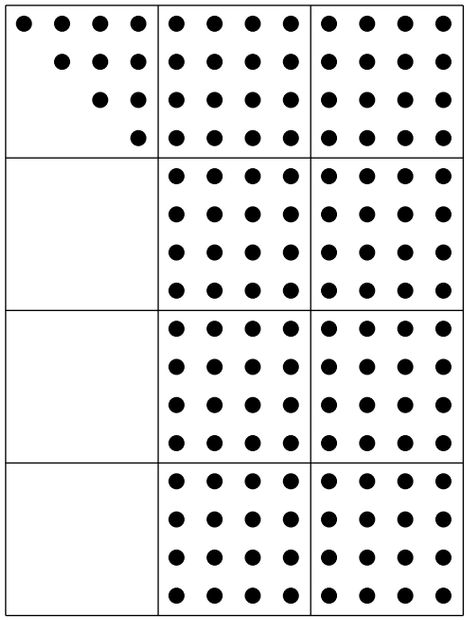
$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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

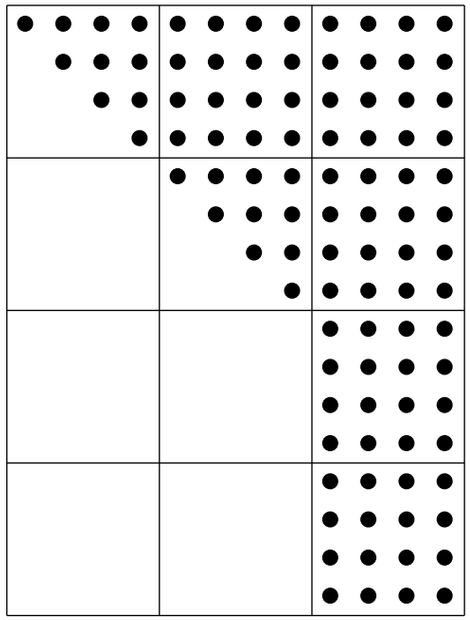
The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



$\mathbf{A}_0 = \mathbf{A}$



$\mathbf{A}_1 = \mathbf{U}_1^* \mathbf{A}_0 \mathbf{V}_1$



$\mathbf{A}_2 = \mathbf{U}_2^* \mathbf{A}_1 \mathbf{V}_2$

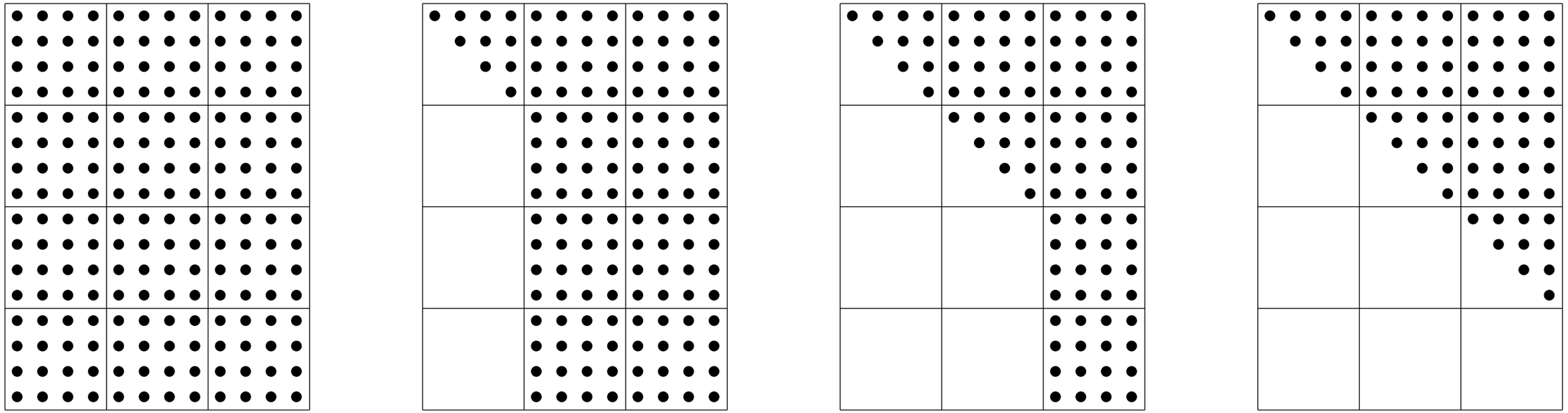
The UTV decomposition: Overview of proposed algorithm randUTV

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

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

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

The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



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

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

Blocking enables high performance. Most flops are spent in matrix-matrix multiplication.

The UTV decomposition: A single blocked step

Consider a single blocked step: We apply unitary matrices $\mathbf{U}$ and $\mathbf{V}$ to get

$$\mathbf{T} = \mathbf{U}^*\mathbf{A}\mathbf{V}.$$

Let b be a block size, and separate out the first b rows and columns so that

$$\mathbf{T} = \begin{bmatrix} \mathbf{U}_1^* \\ \mathbf{U}_2^* \end{bmatrix} \mathbf{A} \begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{0} & \mathbf{T}_{22} \end{bmatrix}.$$

We want the following properties in the transformed matrix $\mathbf{T}$:

- $\mathbf{T}_{11}$ should hold as *much* mass as possible.
- $\mathbf{T}_{12}$ should be tiny.

A *perfect* choice of $\mathbf{U}$ and $\mathbf{V}$ would be:

- The columns of $\mathbf{U}_1$ span the space spanned by the first k left singular vectors.
- The columns of $\mathbf{V}_1$ span the space spanned by the first k right singular vectors.

We use randomization to cheaply find a *close to optimal* choice:

$$\mathbf{V}_1 = (\mathbf{A}^*\mathbf{A})^q \mathbf{A}^*\mathbf{G},$$

where $\mathbf{G}$ is an $n \times b$ Gaussian random matrix, and where $q \in \{0, 1, 2\}$.

(Over-sampling can be used as well.)

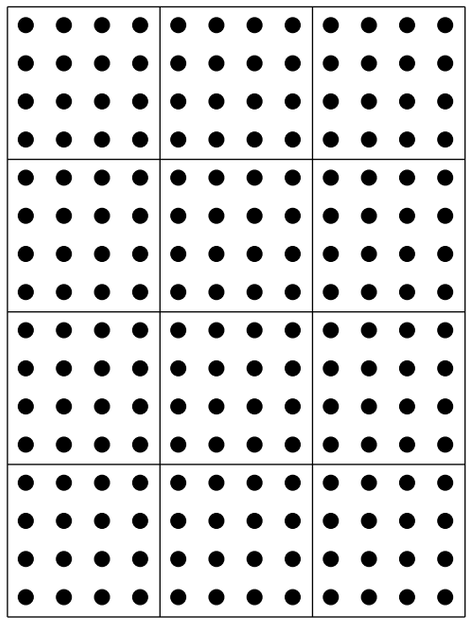
The UTV decomposition: Overview of proposed algorithm randUTV

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

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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

The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



$$\mathbf{A}_0 = \mathbf{A}$$

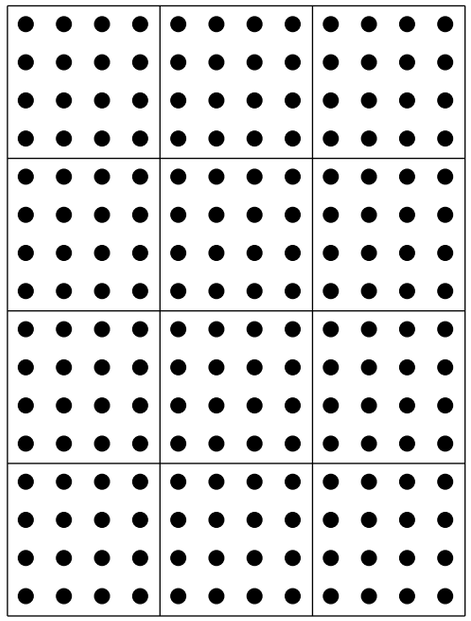
The UTV decomposition: Overview of proposed algorithm randUTV

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

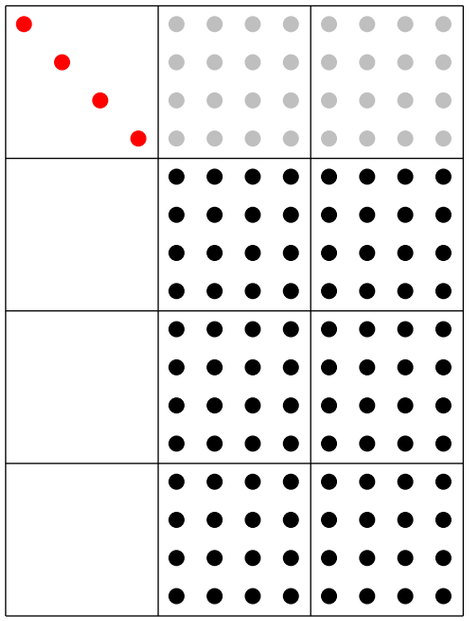
$$\underset{m \times n}{\mathbf{A}} = \underset{m \times n}{\mathbf{U}} \quad \underset{n \times n}{\mathbf{T}} \quad \underset{n \times n}{\mathbf{V}^*},$$

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

The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



$\mathbf{A}_0 = \mathbf{A}$



$\mathbf{A}_1 = \mathbf{U}_1^* \mathbf{A}_0 \mathbf{V}_1$

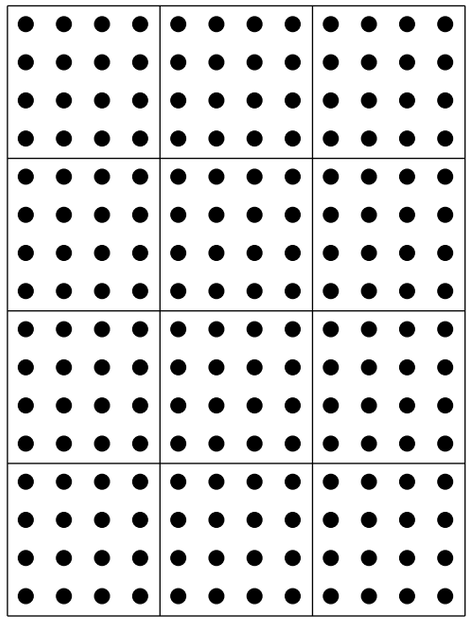
The UTV decomposition: Overview of proposed algorithm randUTV

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

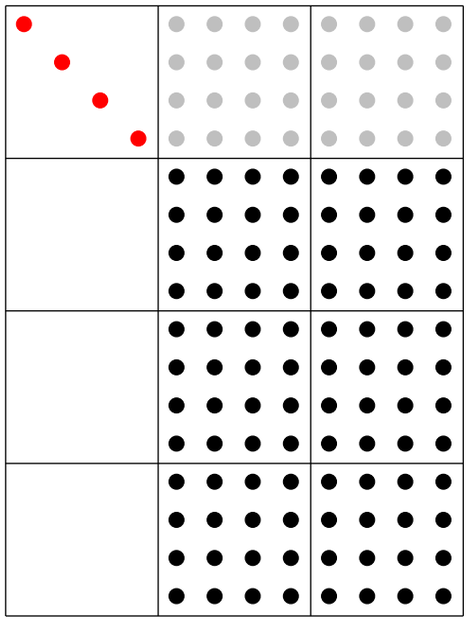
$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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

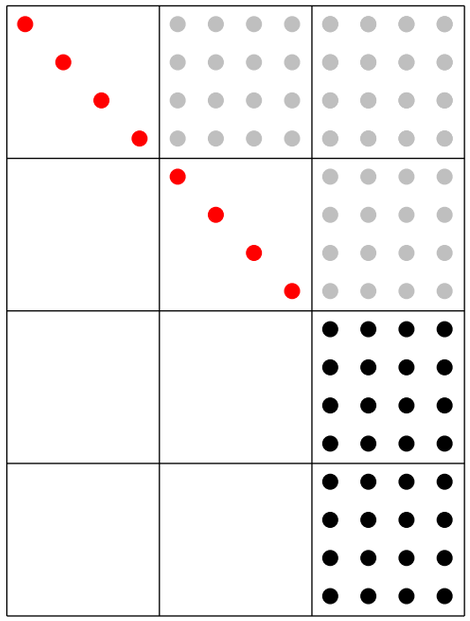
The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



$\mathbf{A}_0 = \mathbf{A}$



$\mathbf{A}_1 = \mathbf{U}_1^* \mathbf{A}_0 \mathbf{V}_1$



$\mathbf{A}_2 = \mathbf{U}_2^* \mathbf{A}_1 \mathbf{V}_2$

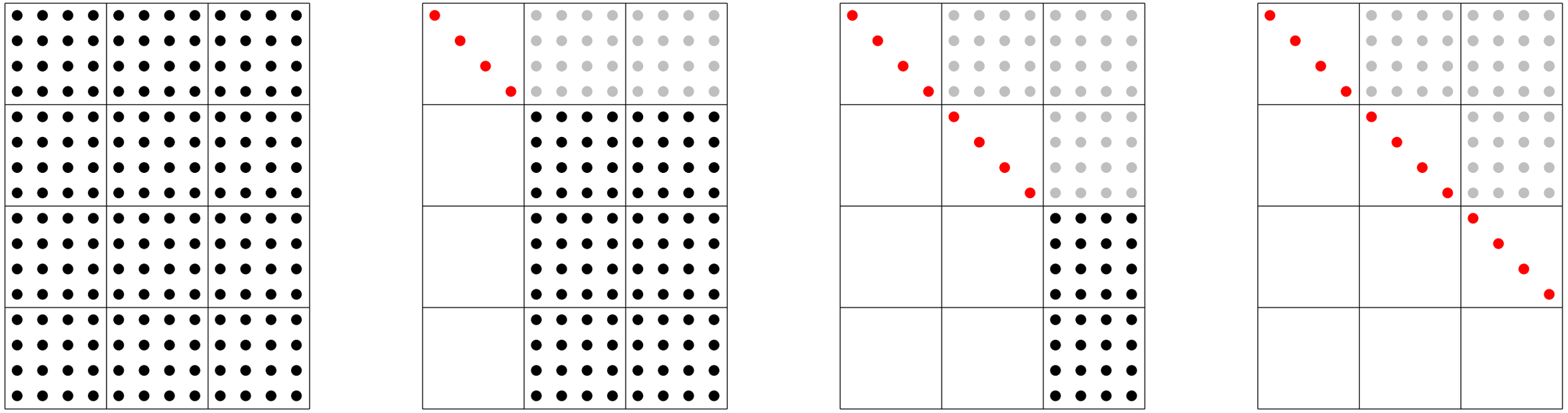
The UTV decomposition: Overview of proposed algorithm randUTV

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

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

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

The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



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

The $\mathbf{V}$ matrices are found using the randomized projections. (Basically RSVD.)

The $\mathbf{U}$ matrices zero out the sub-diagonal elements.

Both $\mathbf{U}$ and $\mathbf{V}$ must be represented efficiently as products of Householder reflectors.

A full, but small (of size $b \times b$) SVD is used to diagonalize the diagonal blocks.

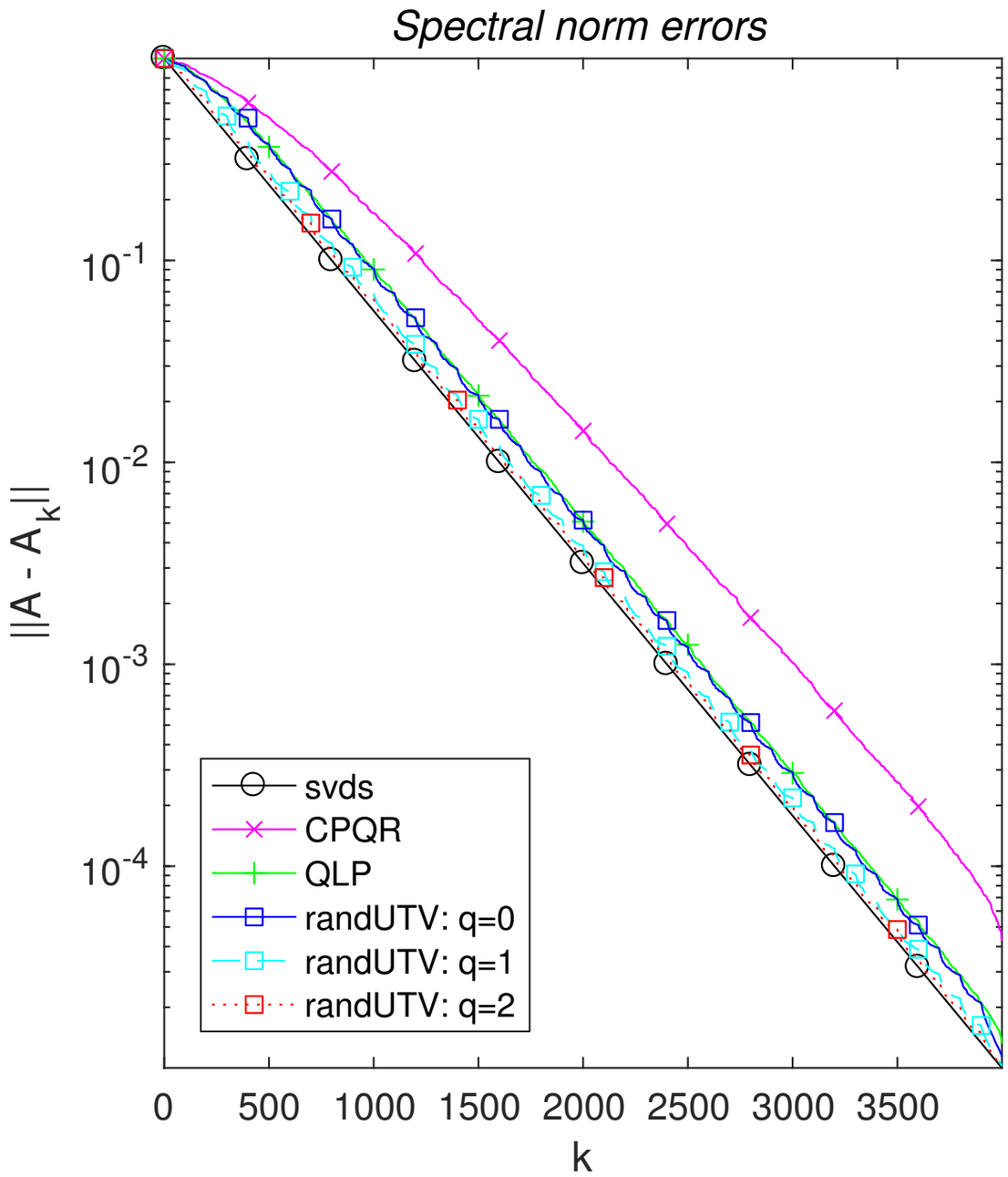
The super-diagonal elements are very small — often of relative size 10^{-5} *or so!*

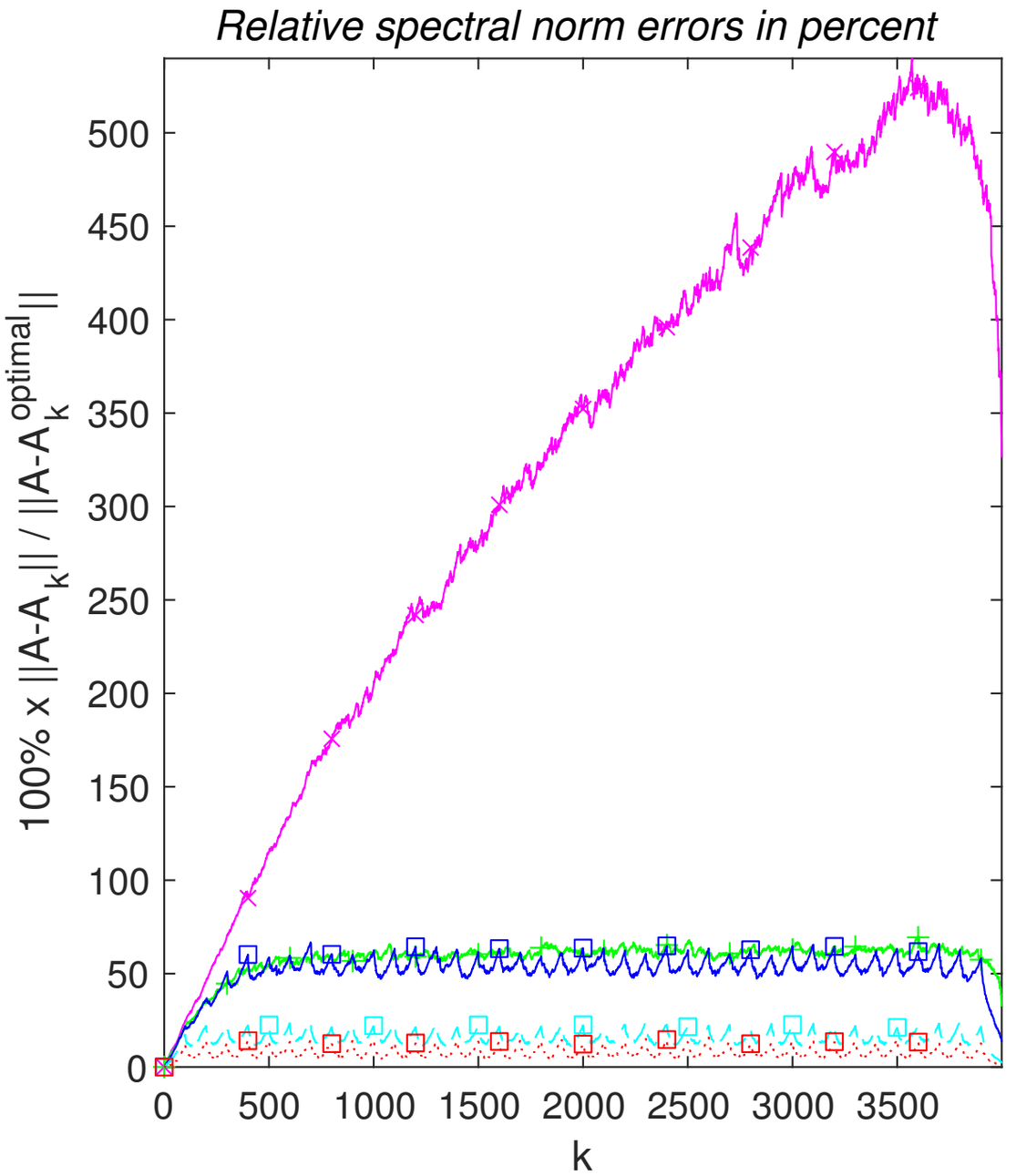
```
function [U,T,V] = randUTV(A,b,q)
  T = A;
  U = eye(size(A,1));
  V = eye(size(A,2));
  for i = 1:ceil(size(A,2)/b)
    I1 = 1:(b*(i-1));
    I2 = (b*(i-1)+1):size(A,1);
    J2 = (b*(i-1)+1):size(A,2);
    if (length(J2) > b)
      [UU,TT,VV] = stepUTV(T(I2,J2),b,q);
    else
      [UU,TT,VV] = svd(T(I2,J2));
    end
    U(:,I2)  = U(:,I2)*UU;
    V(:,J2)  = V(:,J2)*VV;
    T(I2,J2) = TT;
    T(I1,J2) = T(I1,J2)*VV;
  end
return
```

```
function [U,T,V] = stepUTV(A,b,q)
  G = randn(size(A,1),b);
  Y = A'*G;
  for i = 1:q
    Y = A'*(A*Y);
  end
  [V,~]    = qr(Y);
  [U,D,W]  = svd(A*V(:,1:b));
  T        = [D,U'*A*...
                V(:,(b+1):end)];
  V(:,1:b) = V(:,1:b)*W;
return
```

Matlab code for the algorithm randUTV that given an $m \times n$ matrix $\mathbf{A}$ *computes its UTV factorization* $\mathbf{A} = \mathbf{UTV}^*$*. The input parameters b and q reflect the block size and the number of steps of power iteration, respectively. In actual implementations, all unitary matrices are stored as products of Householder reflectors.*

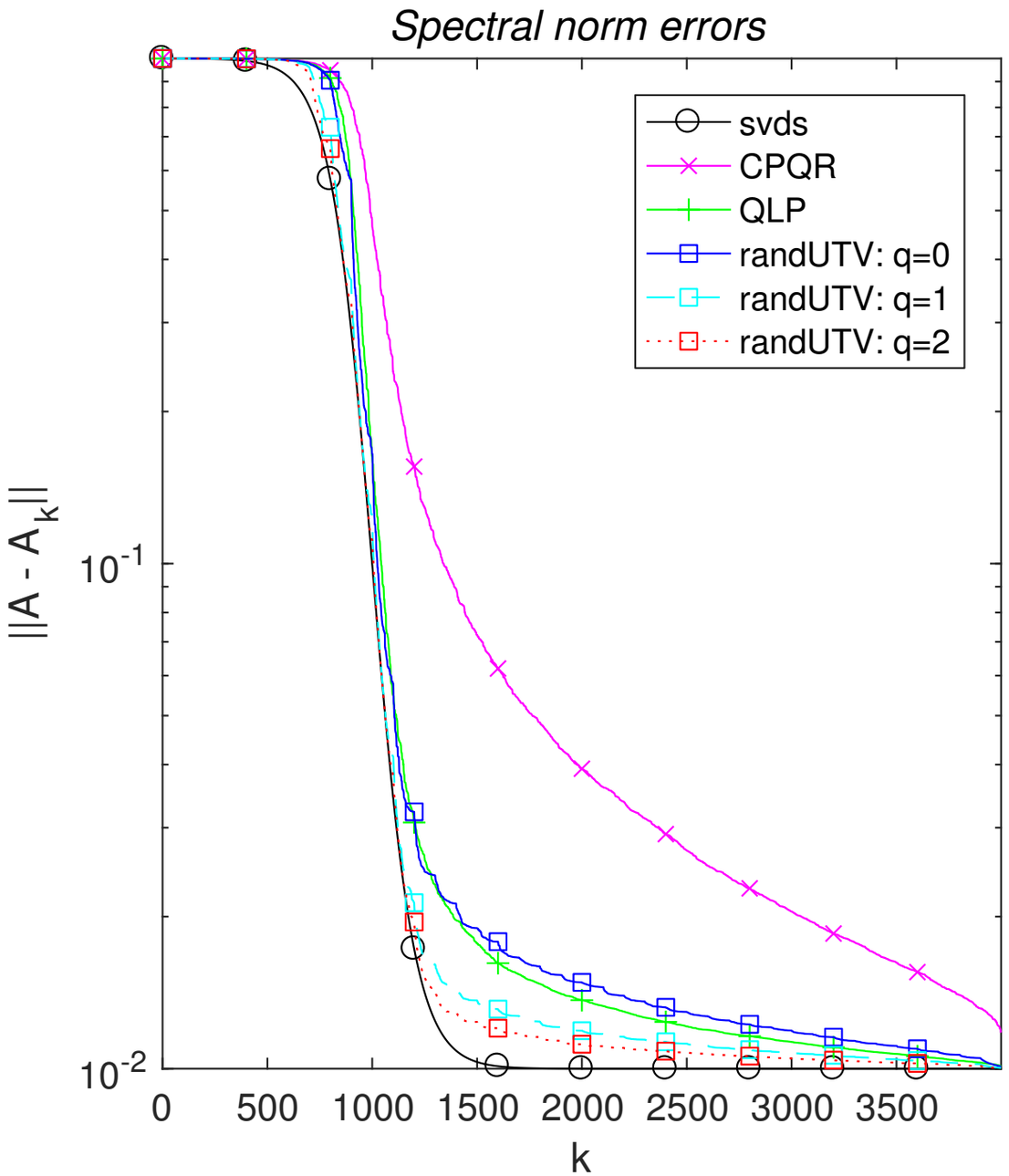
Numerical experiments illustrating the errors in the UTV factorization

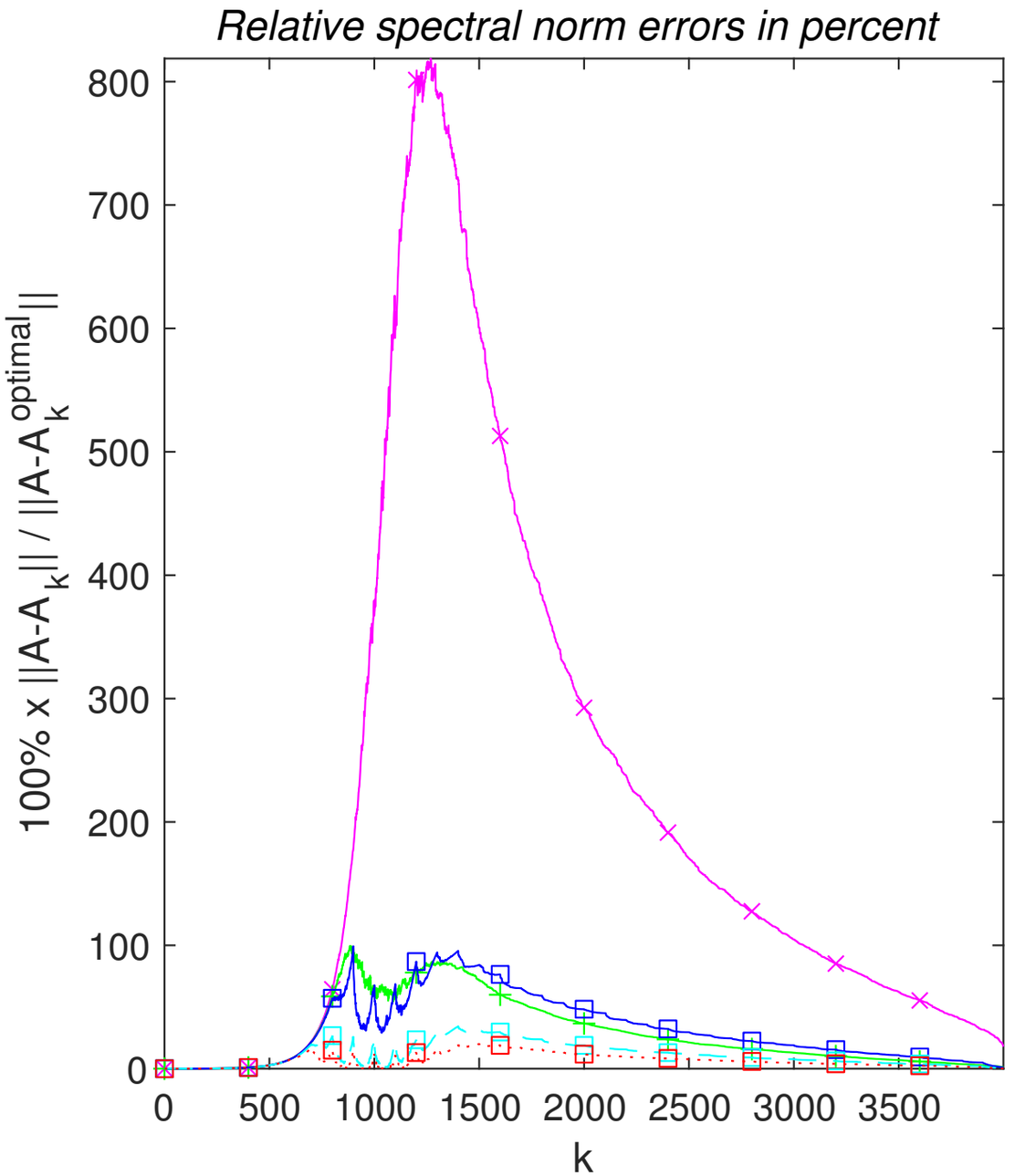




Rank-k approximation errors for the matrix "Fast Decay" of size 4000×4000. *The block size was* $b = 100$. *Left: Absolute errors in spectral norm. The black line (circles) marks the theoretically minimal errors. Right: Relative errors* $e_k^{\text{relative}} = 100\% \times \dfrac{\|\mathbf{A} - \mathbf{A}_k\|}{\|\mathbf{A} - \mathbf{A}_k^{\text{optimal}}\|}$.

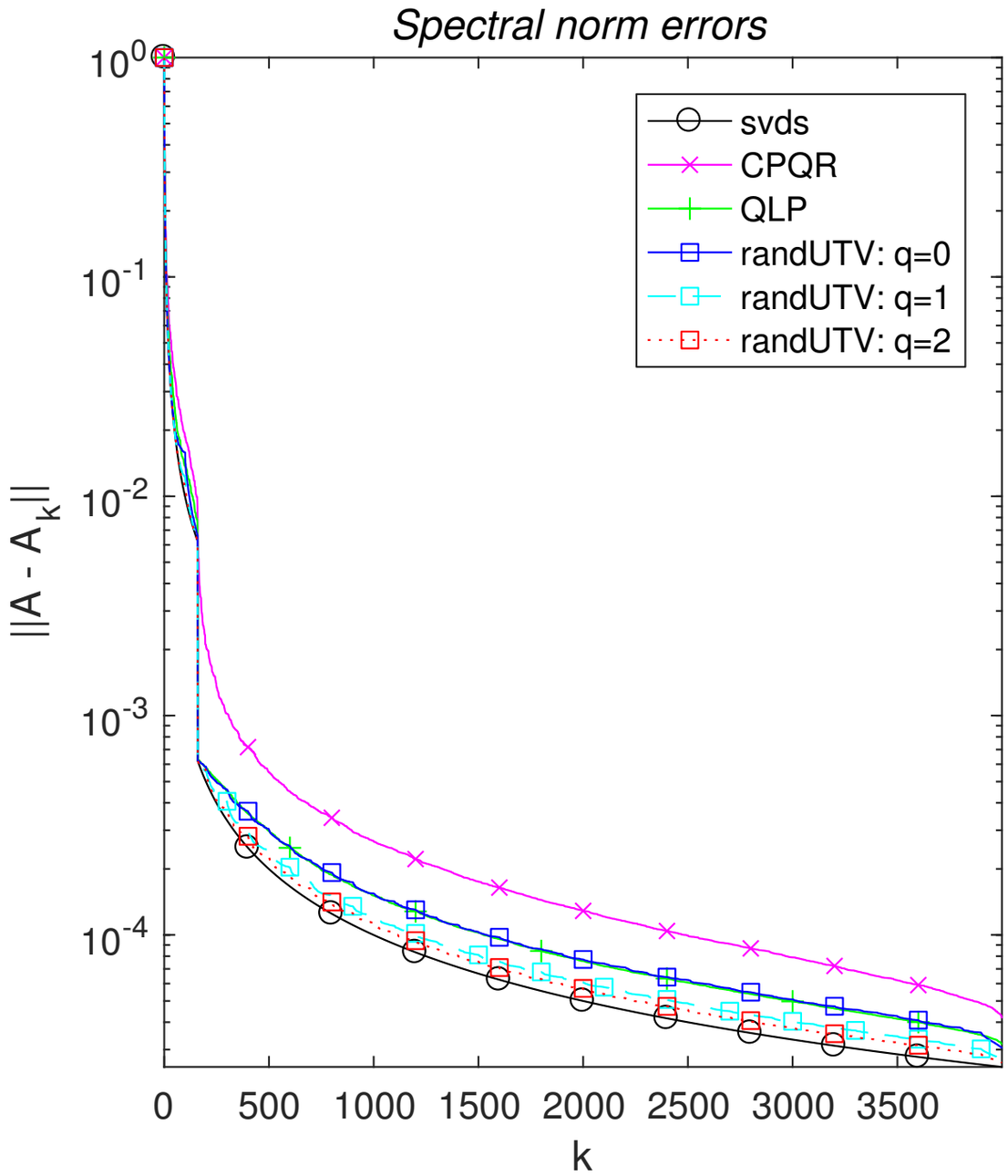
Numerical experiments illustrating the errors in the UTV factorization

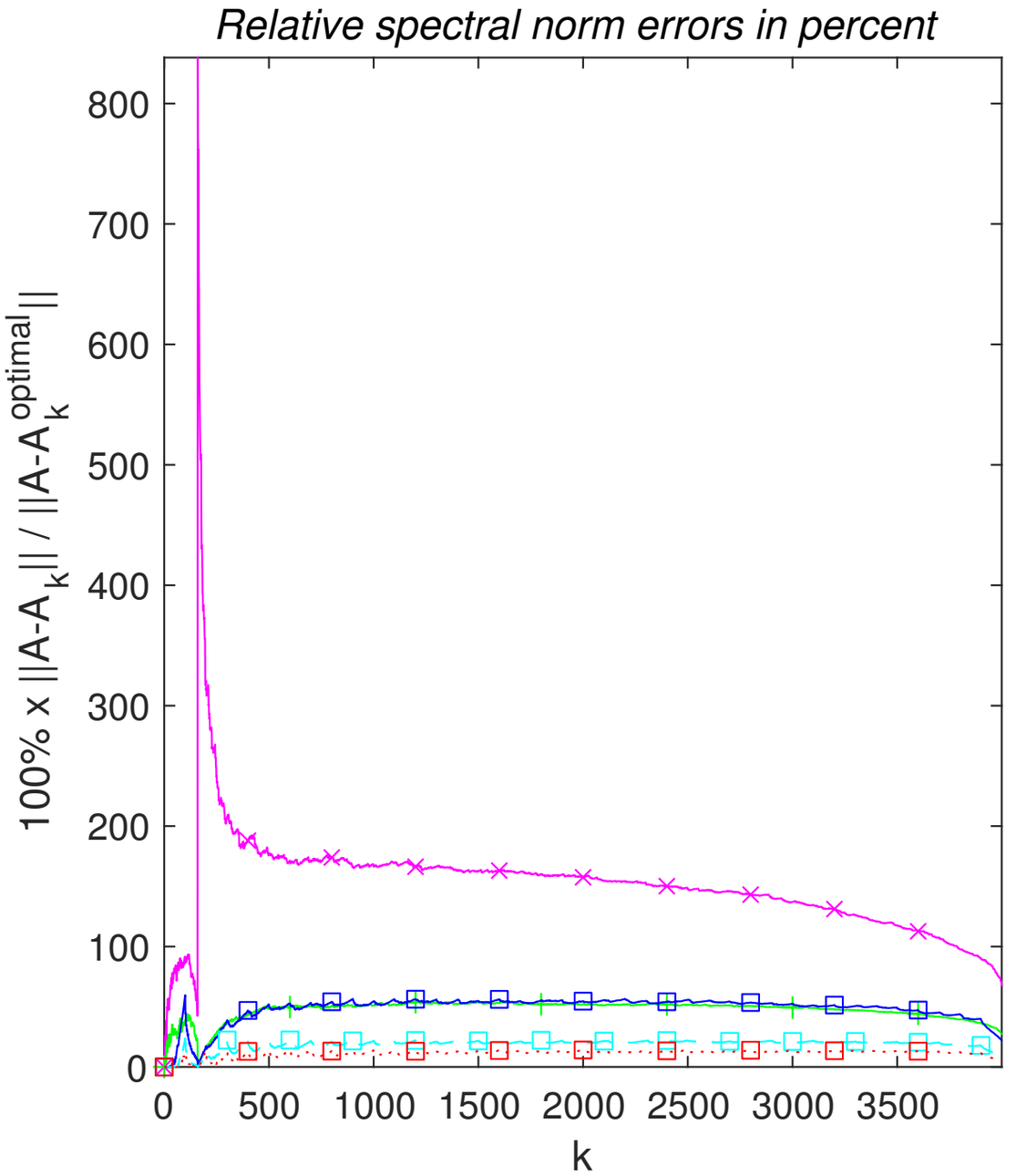




Rank-k approximation errors for the matrix "S-shape" of size 4000×4000. *The block size was* $b = 100$. *Left: Absolute errors in spectral norm. The black line (circles) marks the theoretically minimal errors. Right: Relative errors* $e_k^{\text{relative}} = 100\% \times \dfrac{\|\mathbf{A} - \mathbf{A}_k\|}{\|\mathbf{A} - \mathbf{A}_k^{\text{optimal}}\|}$.

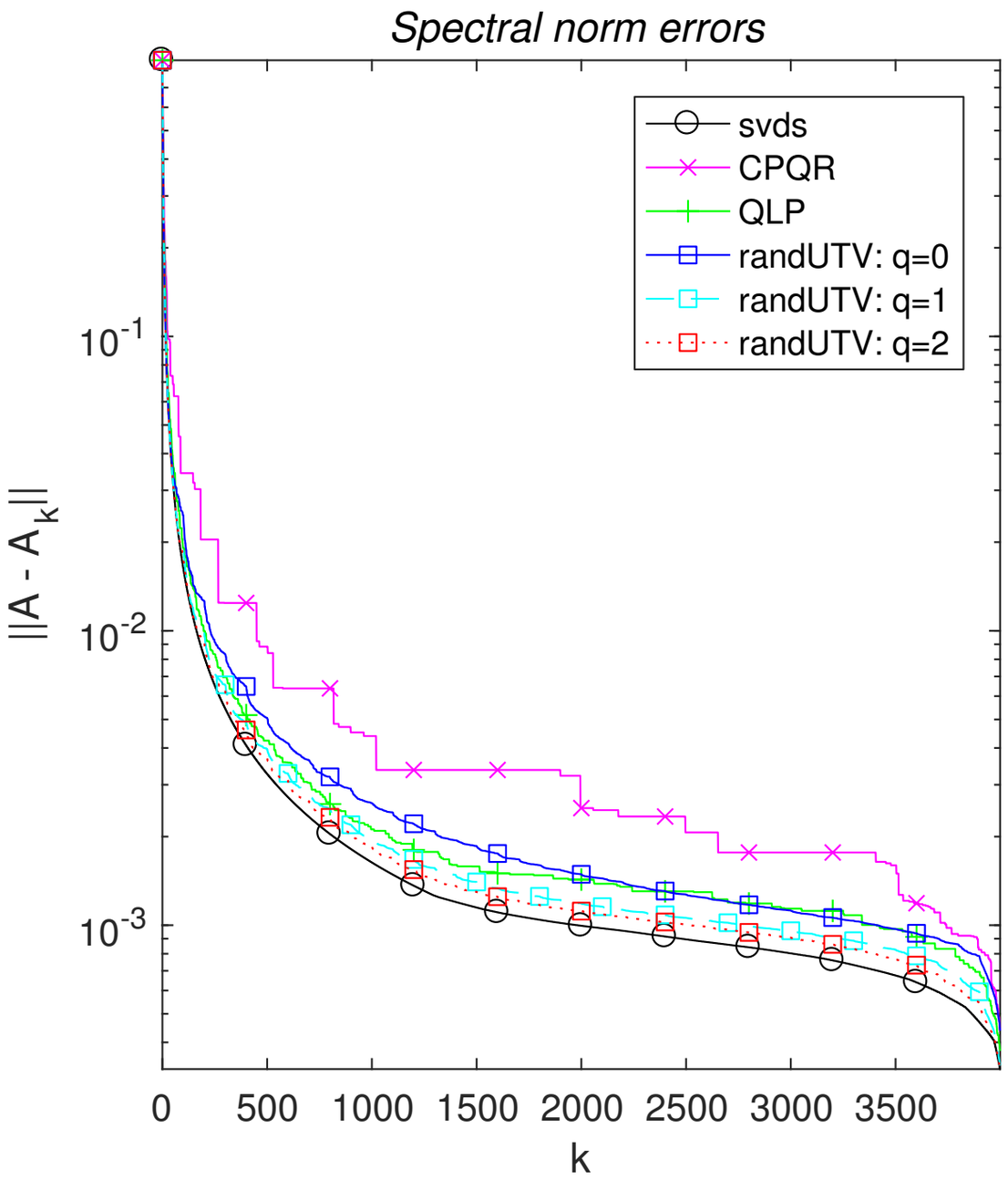
Numerical experiments illustrating the errors in the UTV factorization

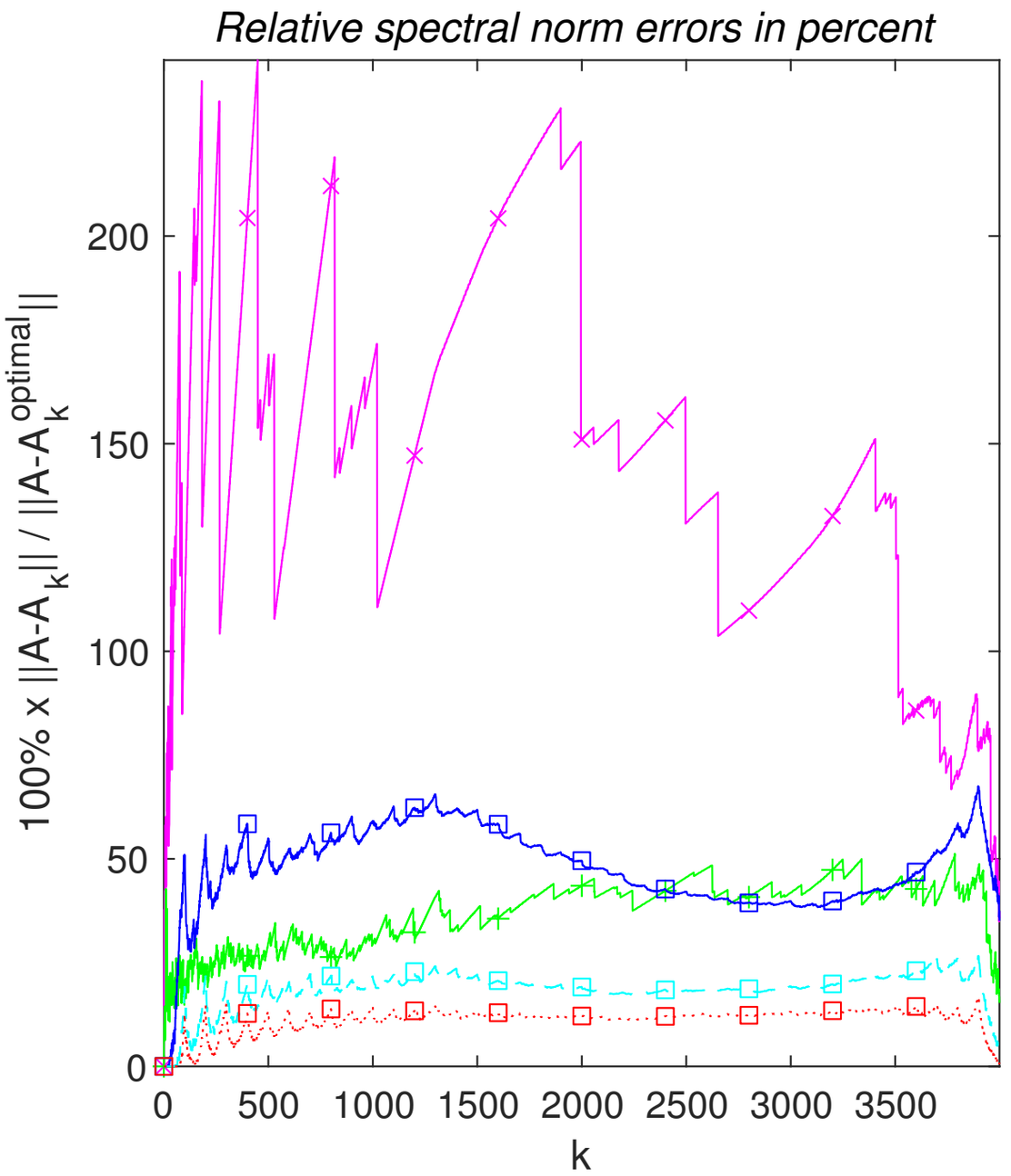




Rank-k approximation errors for the matrix "Gap" of size 4000×4000. *The block size was* $b = 100$. *Left: Absolute errors in spectral norm. The black line (circles) marks the theoretically minimal errors. Right: Relative errors* $e_k^{\text{relative}} = 100\% \times \dfrac{\|\mathbf{A} - \mathbf{A}_k\|}{\|\mathbf{A} - \mathbf{A}_k^{\text{optimal}}\|}$.

Numerical experiments illustrating the errors in the UTV factorization





Rank-k approximation errors for the matrix "BIE" of size 4000×4000. *The block size was* $b = 100$. *Left: Absolute errors in spectral norm. The black line (circles) marks the theoretically minimal errors. Right: Relative errors* $e_k^{\text{relative}} = 100\% \times \dfrac{\|\mathbf{A} - \mathbf{A}_k\|}{\|\mathbf{A} - \mathbf{A}_k^{\text{optimal}}\|}$.

Numerical experiments illustrating how close the UTV is to the SVD

As a consequence of the fact that the super-diagonal elements of **T** are very small, the diagonal elements of **T** are excellent approximants to the singular values of **A**:

$$\mathbf{T}(j,j) \approx \sigma_j, \qquad j = 1,\, 2\, \ldots,\, \min(m,n).$$

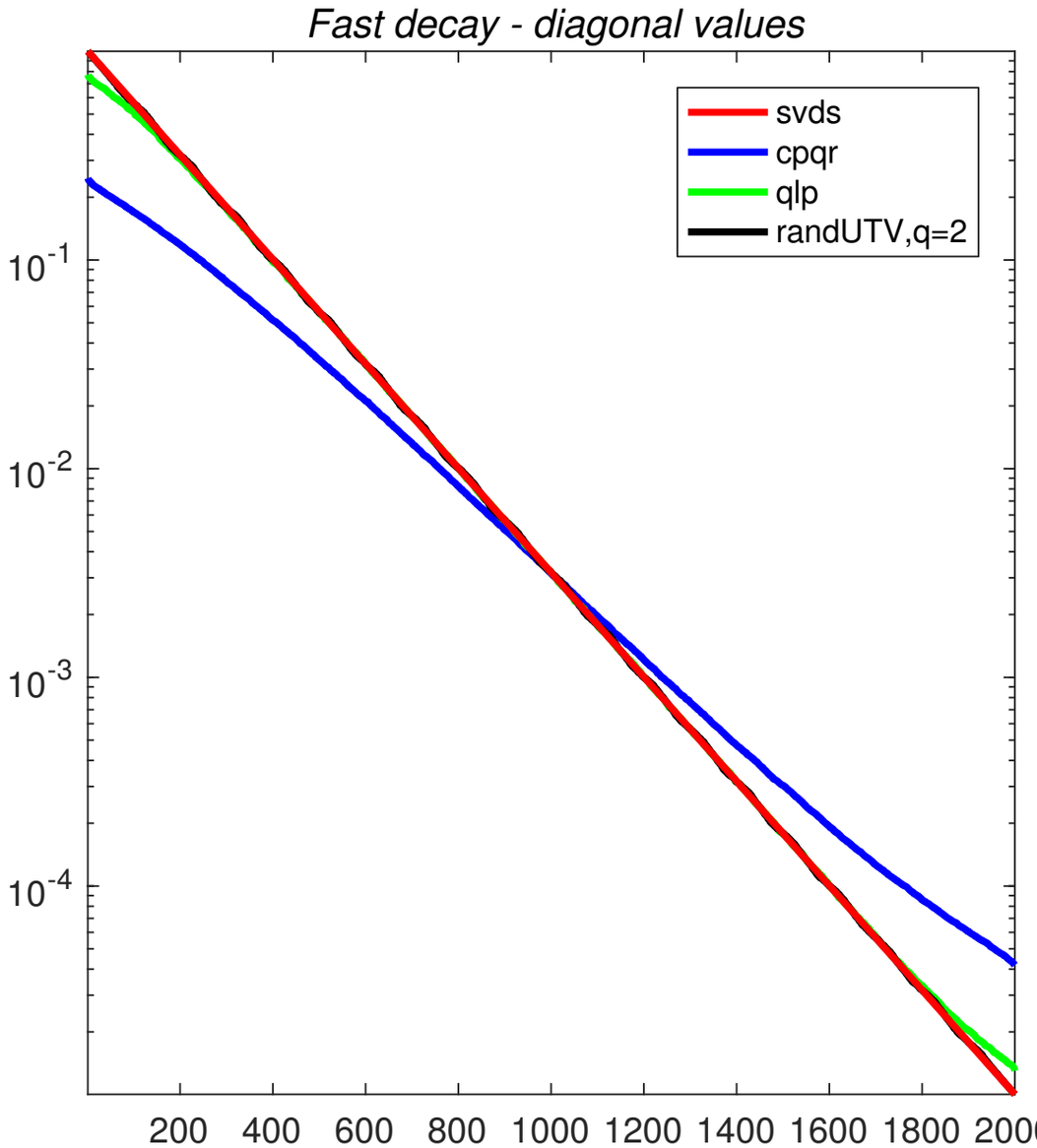
Question: How good?

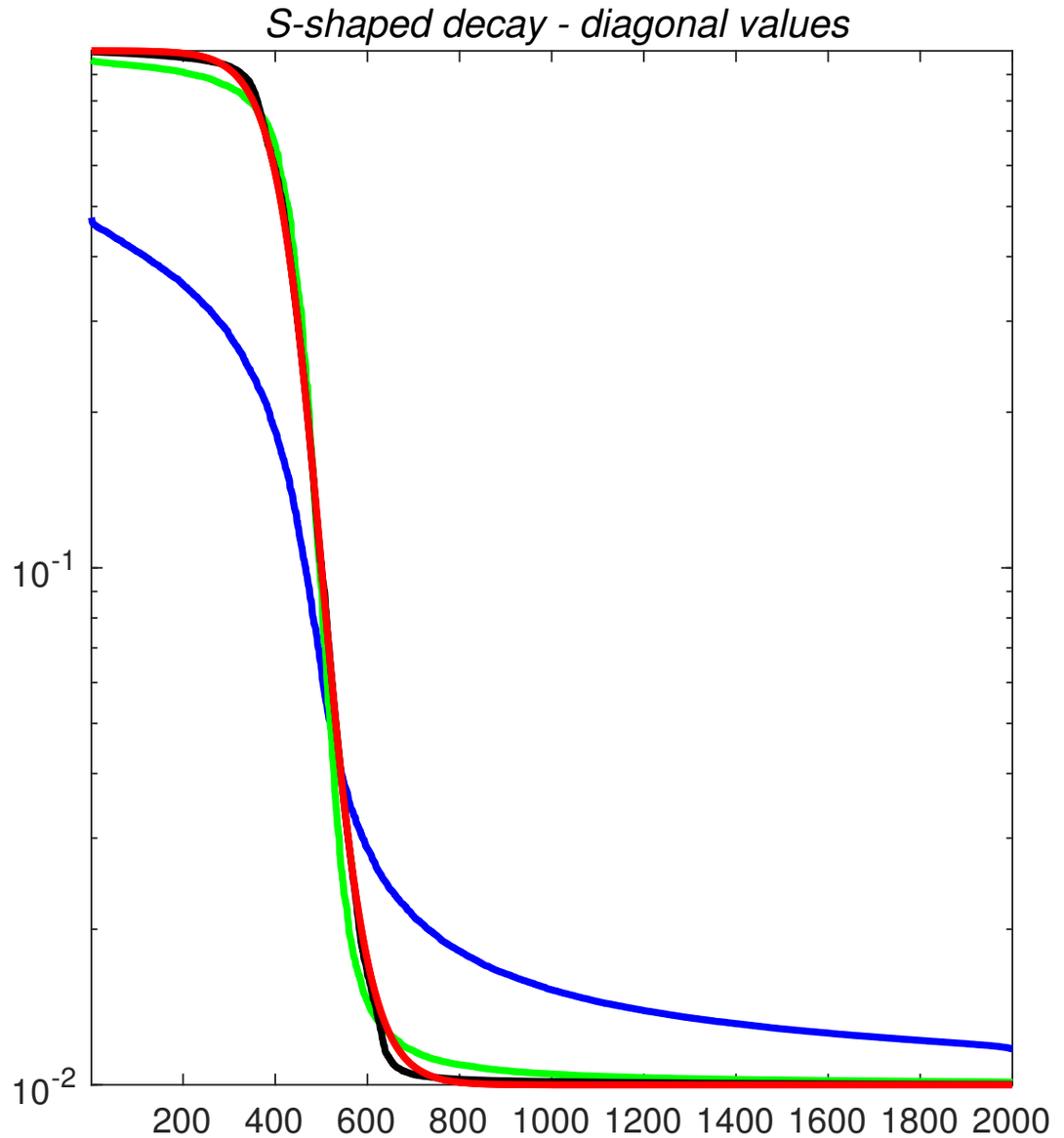
Numerical experiments illustrating how close the UTV is to the SVD

As a consequence of the fact that the super-diagonal elements of **T** are very small, the diagonal elements of **T** are excellent approximants to the singular values of **A**:

$$\mathbf{T}(j,j) \approx \sigma_j, \qquad j = 1,\, 2\, \ldots,\, \min(m,n).$$

Question: How good?



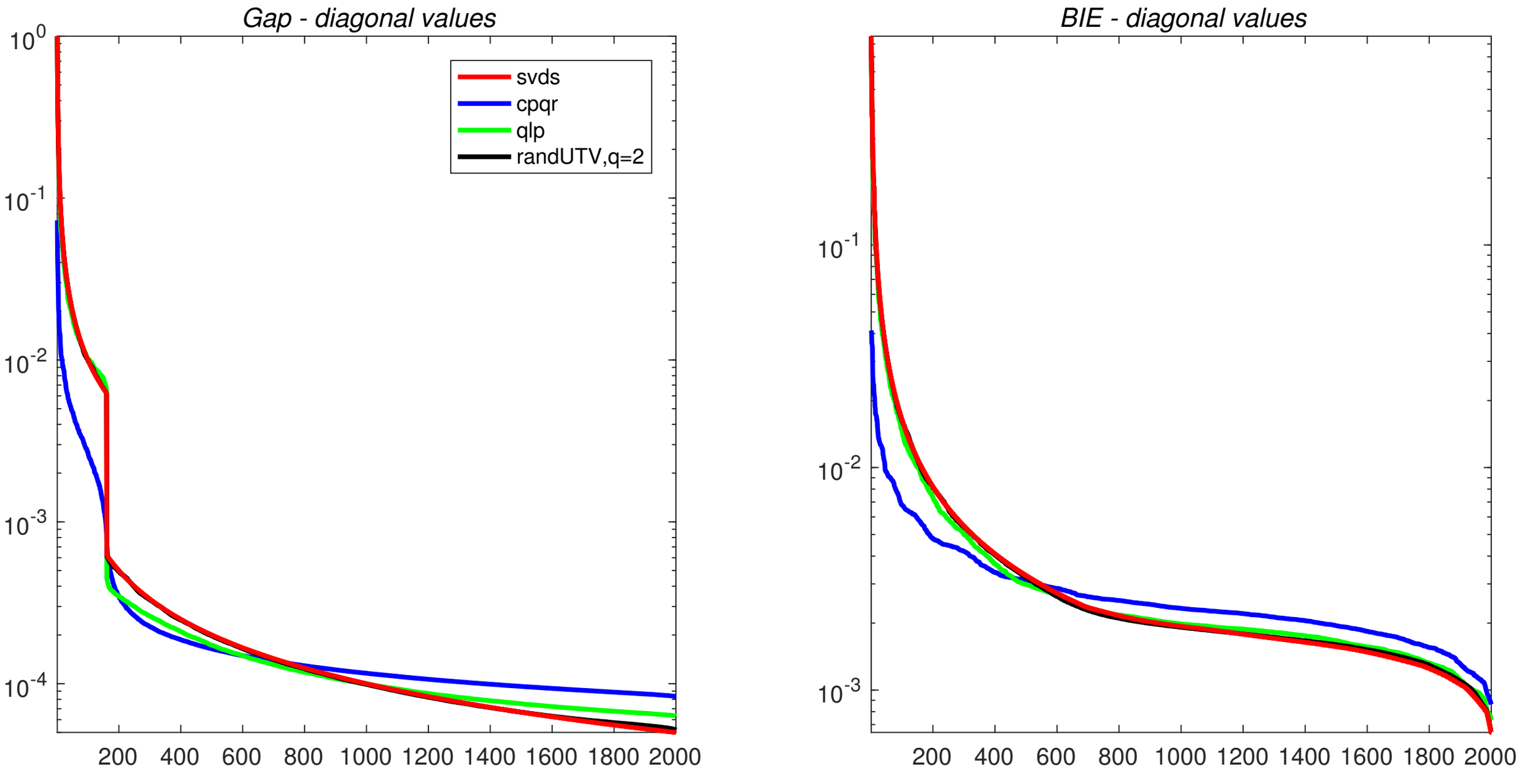


Numerical experiments illustrating how close the UTV is to the SVD

As a consequence of the fact that the super-diagonal elements of **T** are very small, the diagonal elements of **T** are excellent approximants to the singular values of **A**:

$$\mathbf{T}(j,j) \approx \sigma_j, \qquad j = 1,\, 2\, \ldots,\, \min(m,n).$$

Question: How good?



Orthonormal matrices (14 cores)

$10^{10} \times \text{Time} / n^3$

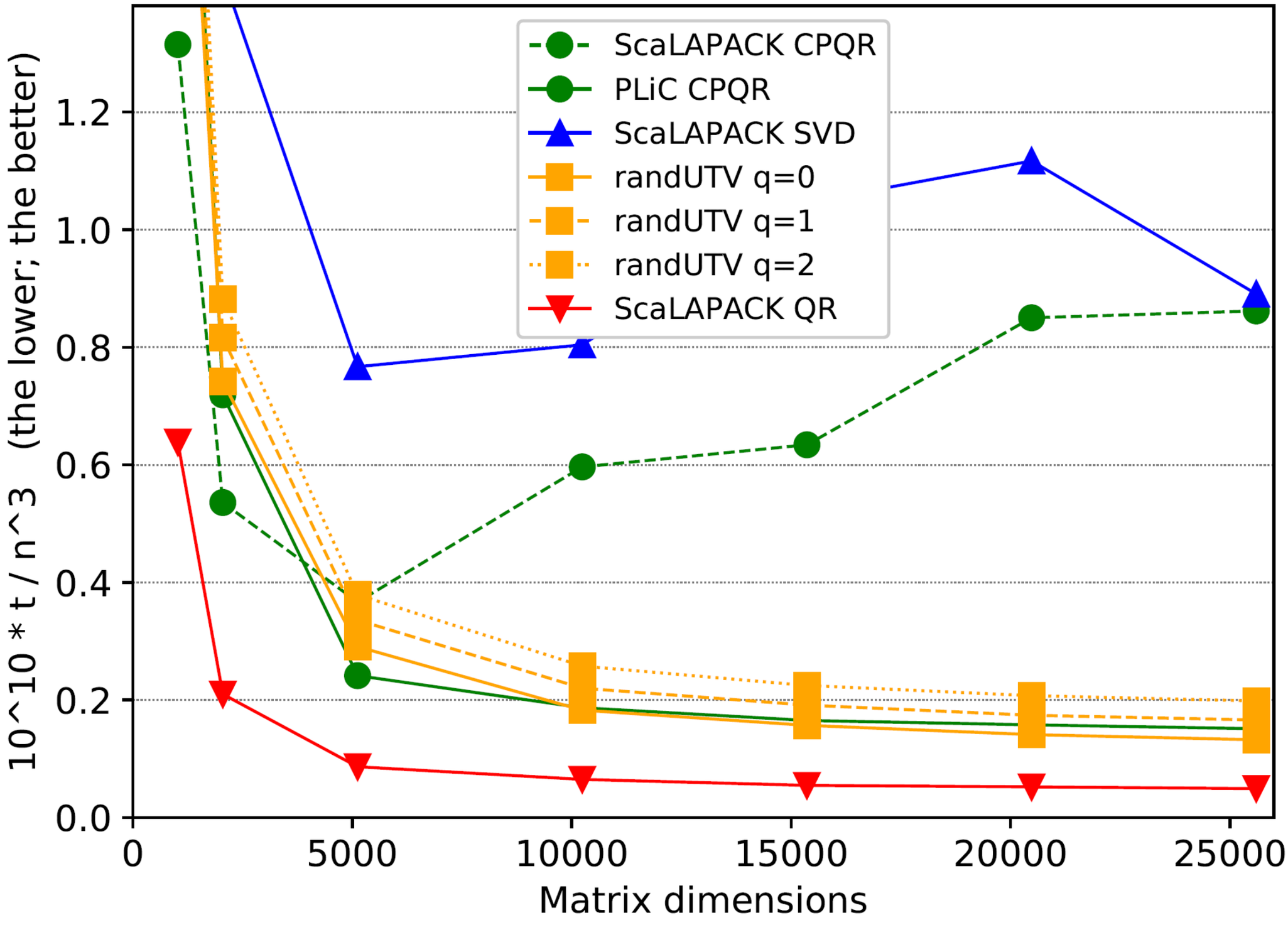
2
1.5
1
0.5
0

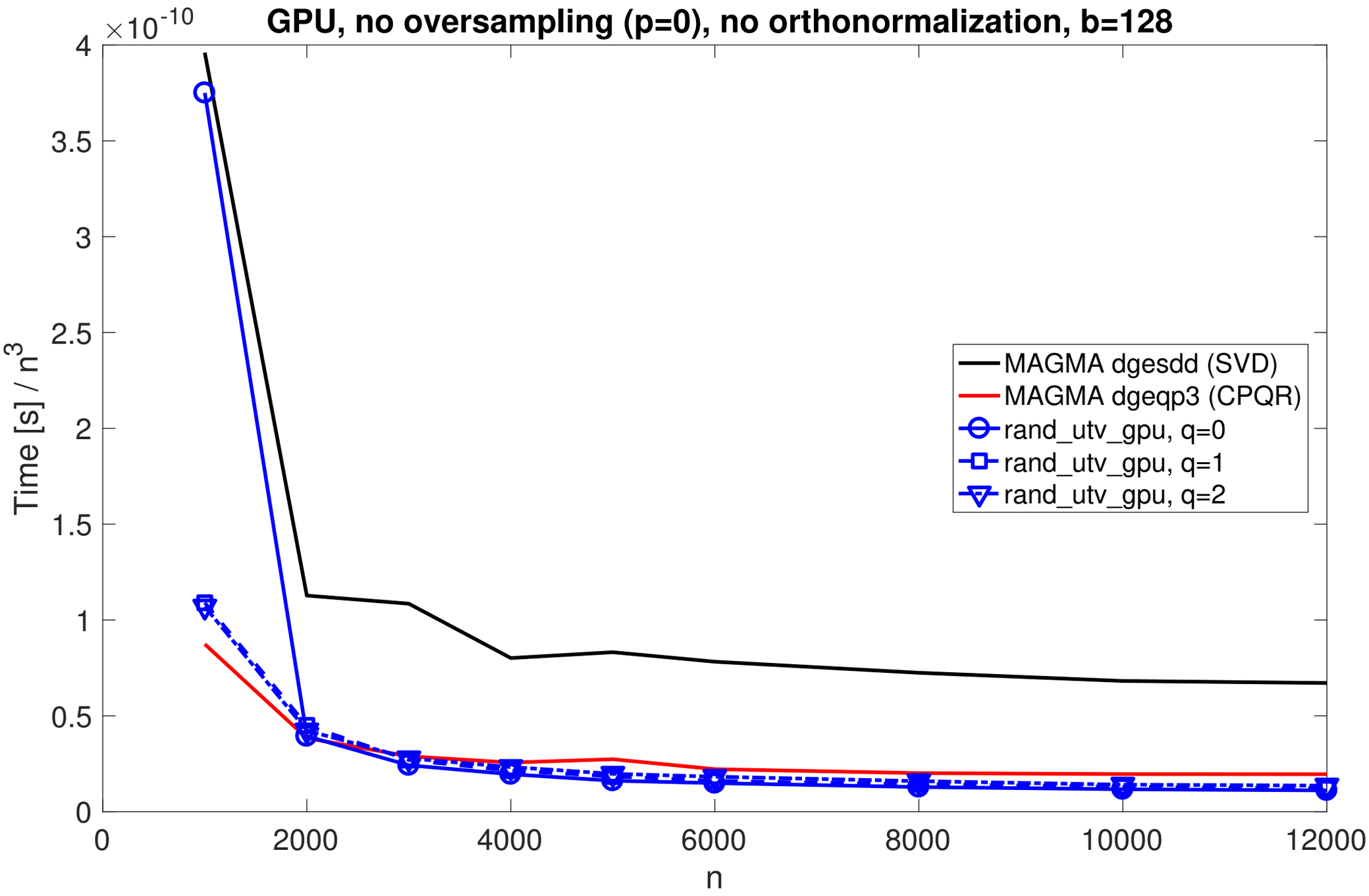
0
20480
40960
61440
81920
102400

n

In-core MKL SVD
In-core MKL CPQR
In-core randUTV pblas q=2
Out-of-core randUTV v21v q=2

Performances on a 6 x 16 mesh. Orthonormal matrices are built.




GPU, no oversampling (p=0), no orthonormalization, b=128
×10^-10
Time [s] / n^3
n
0
0.5
1
1.5
2
2.5
3
3.5
4
0
2000
4000
6000
8000
10000
12000
MAGMA dgesdd (SVD)
MAGMA dgeqp3 (CPQR)
rand_utv_gpu, q=0
rand_utv_gpu, q=1
rand_utv_gpu, q=2

Randomized Column Pivoted QR (randCPQR)

Given an $m \times n$ matrix $\mathbf{A}$ (with $m \geq n$), we seek a QR factorization

$$\begin{array}{cccc} \mathbf{A} & \mathbf{P} & \approx & \mathbf{Q} \quad \mathbf{R} \\ m \times n & n \times n & & m \times k \;\; k \times n \end{array}$$

for either $k = n$ (full factorization) or k comparable to $\min(m, n)$. As usual, $\mathbf{Q}$ is orthonormal, $\mathbf{P}$ is a permutation, and $\mathbf{R}$ is upper triangular.

Question: Is the CPQR "rank-revealing"? Does it satisfy:

- The truncated factorization is a close to optimal low-rank factorization, so that

$$\|\mathbf{A} - \mathbf{Q}(:, 1:k)\,\mathbf{R}(1:k, :)\mathbf{P}^*\| =\approx \inf\{\|\mathbf{A} - \mathbf{B}\| : \ \mathbf{B} \text{ has rank } k\}.$$

- $\sigma_j(\mathbf{T}(1:k, 1:k)) \approx \sigma_j(\mathbf{A})$ for $j \in \{1, 2, \ldots, k\}$.

In practice, it is pretty good; it is often used as a cheap substitute for SVD.

There are counter-examples, for which it performs very badly.

Note: There are sophisticated pivoting strategies that improve on how well CPQR reveals numerical rank — seminal work by Gu and Eisenstat (1996). Tricky to implement efficiently.

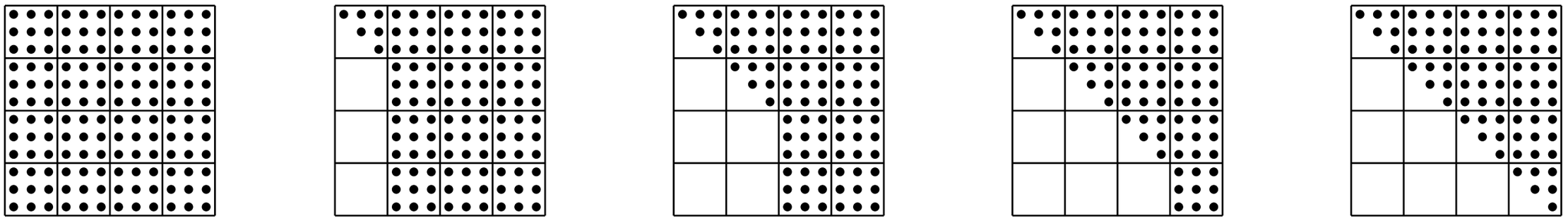
Randomized Column Pivoted QR (randCPQR)

Given an $m \times n$ matrix $\mathbf{A}$ (with $m \geq n$), we seek a QR factorization

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

for either $k = n$ (full factorization) or k comparable to $\min(m, n)$. As usual, $\mathbf{Q}$ is orthonormal, $\mathbf{P}$ is a permutation, and $\mathbf{R}$ is upper triangular.

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

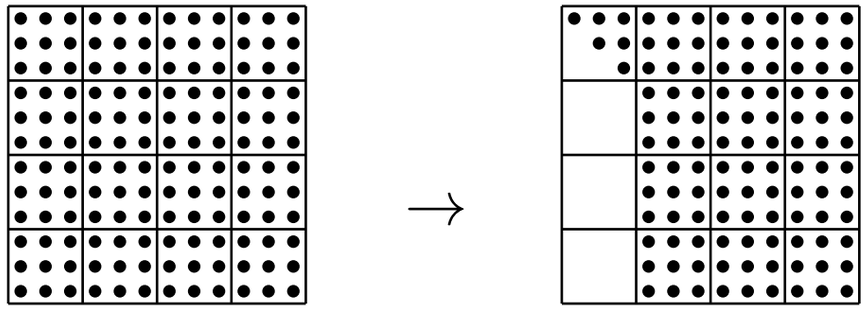


$$\mathbf{A}_0 = \mathbf{A} \qquad \mathbf{A}_1 = \mathbf{Q}_1^* \mathbf{A}_0 \mathbf{P}_1 \qquad \mathbf{A}_2 = \mathbf{Q}_2^* \mathbf{A}_1 \mathbf{P}_2 \qquad \mathbf{A}_3 = \mathbf{Q}_3^* \mathbf{A}_2 \mathbf{P}_3 \qquad \mathbf{A}_4 = \mathbf{Q}_4^* \mathbf{A}_3 \mathbf{P}_4$$

Each $\mathbf{Q}_j$ is a product of Householder reflectors. Each $\mathbf{P}_j$ is a permutation matrix computed via randomized sampling.

Randomized Column Pivoted QR. *How to do block pivoting using randomization:*

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



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

Q is a product of b Householder reflectors. **P** is a pivoting 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

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

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**:

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

References: Martinsson, arxiv, 2015. Martinsson, Quintana-Orti, Heavner, van de Giejn, SISC, 2017. Duersch & Gu, arxiv, 2015. Duersch & Gu, SISC, 2017.

Connection to randomized Interpolatory Decomposition (ID), CUR, etc.

Let **A** be an $m \times n$ matrix of numerical rank k. An *Interpolatory Decomposition (ID)* of **A** takes the form

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{C}} \; \underset{k \times n}{\mathbf{X}}$$

where **C** consists of k columns of **A**, and where **X** is a well-conditioned matrix.

Let J_{S} denote an index vector identifying the "skeleton" columns so that $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{S}})$.

A randomized algorithm for computing the ID, given an over-sampling parameter p:

- Draw a $(k+p) \times m$ Gaussian matrix **G**.
- Form a $(k+p) \times n$ sampling matrix $\mathbf{Y} = \mathbf{GA}$.
- Perform a rank-k CPQR on **Y** so that $\mathbf{Y} \approx \mathbf{Y}(:, J_{\mathrm{S}})\mathbf{X}$.

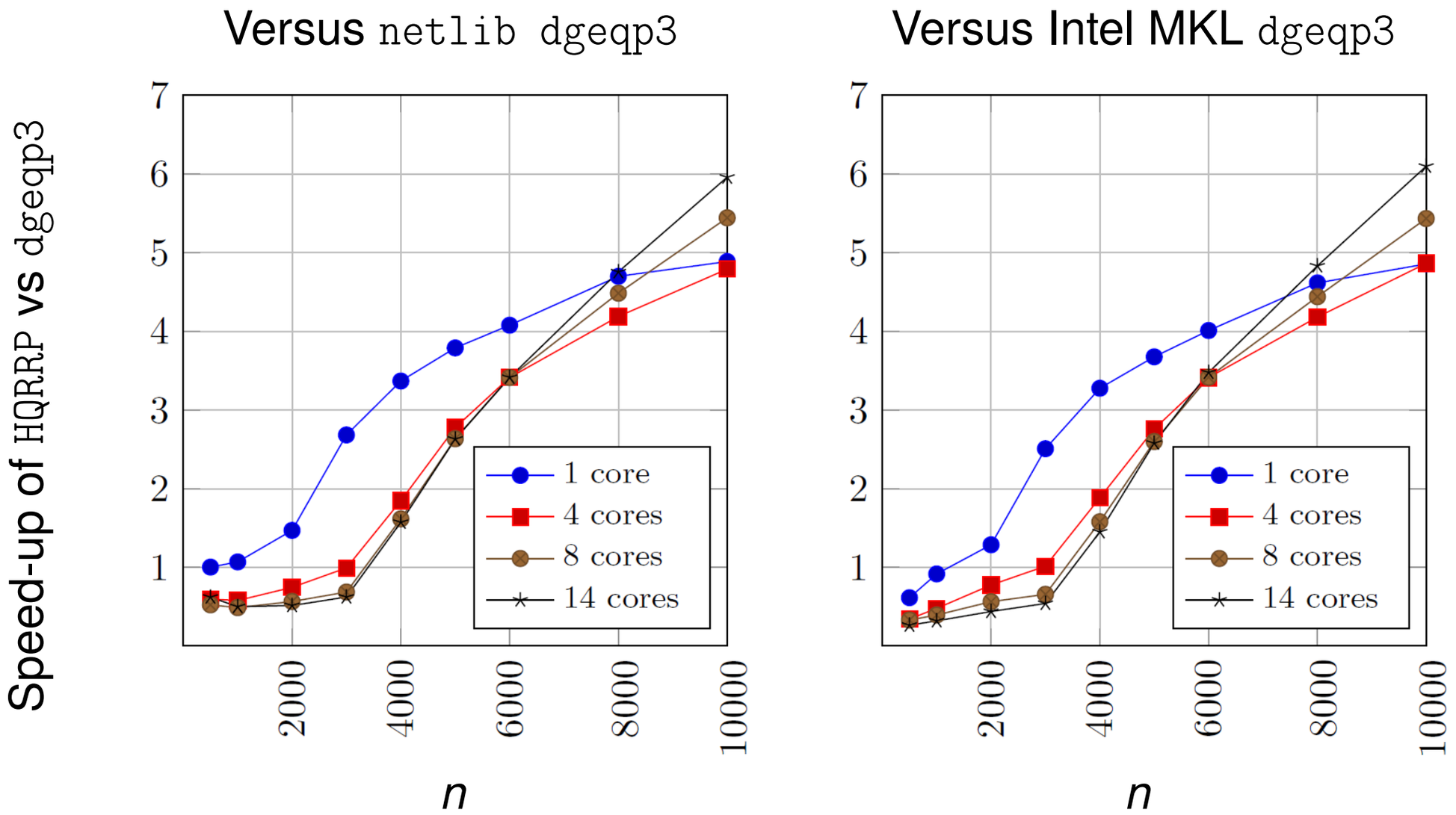
Then we automatically (and almost magically) get an ID of **A**:

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

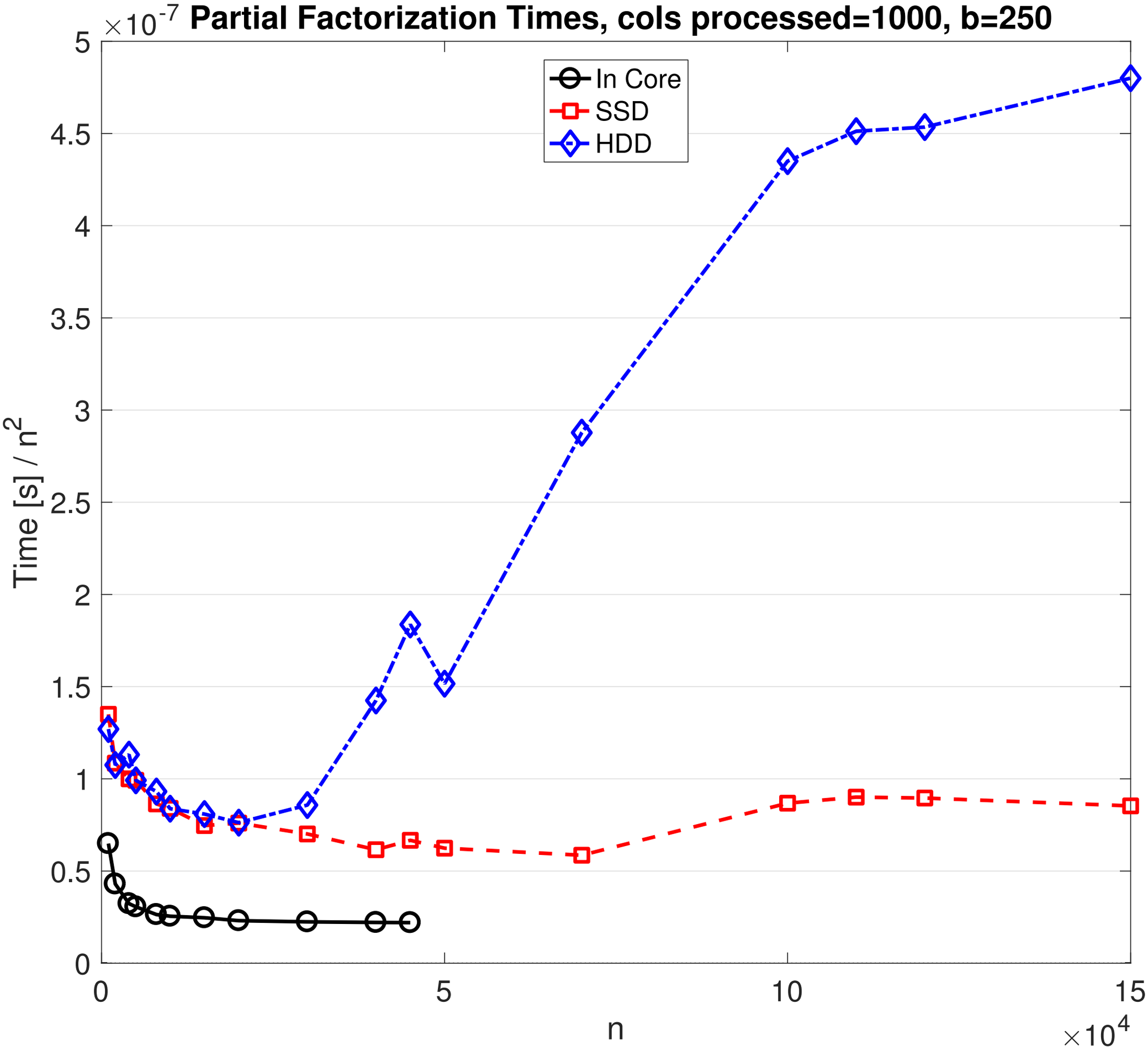
Can be used to compute a CUR decomposition as well.

Reference: "Randomized algorithms for the low-rank approximation of matrices." E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, and M. Tygert; PNAS, 2007

Randomized Column Pivoted QR (randCPQR)

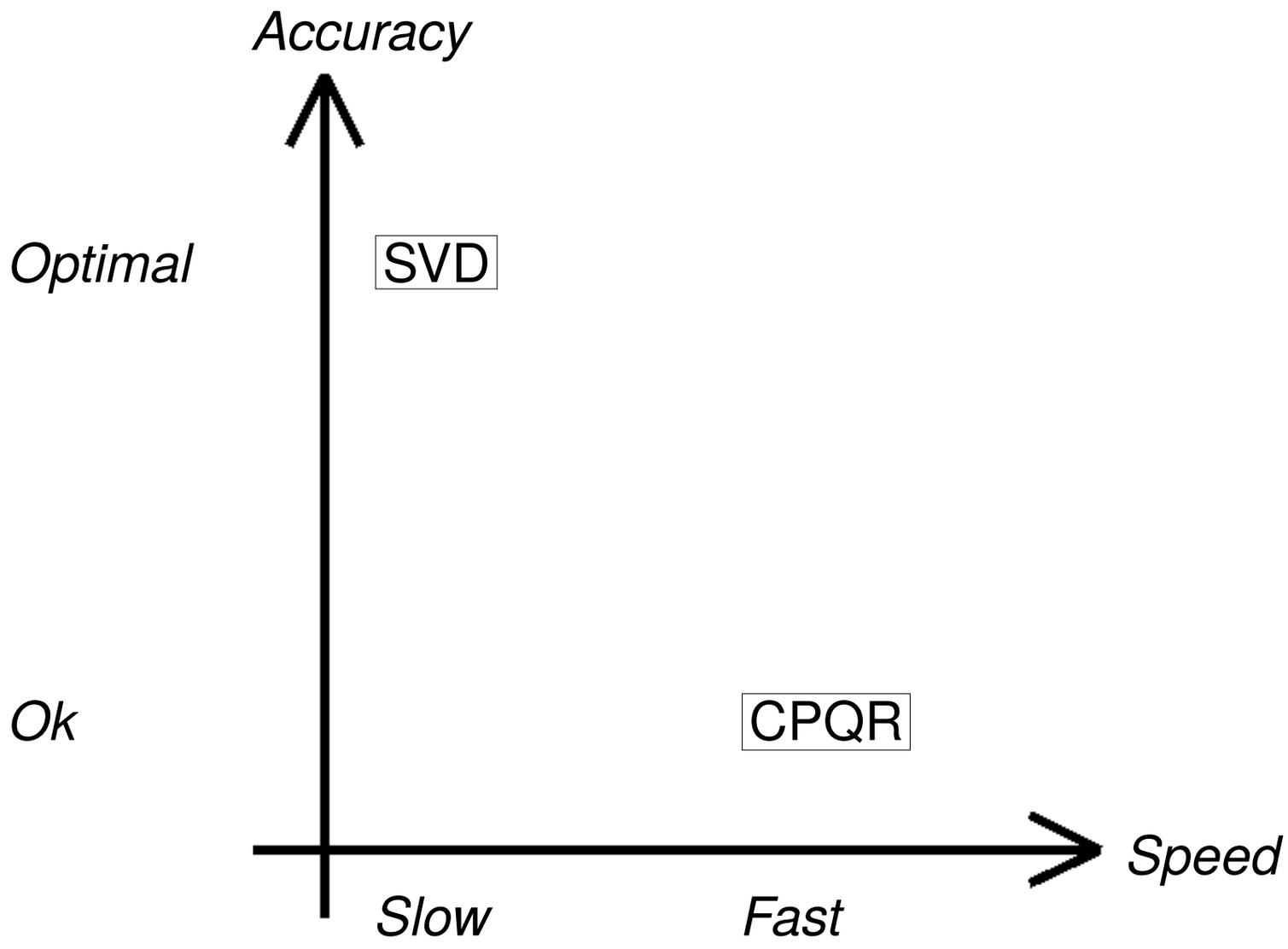


Speedup attained by our randomized algorithm `HQRRP` *for computing a full column pivoted QR factorization of an* $n \times n$ *matrix. The speed-up is measured versus LAPACK's faster routine* `dgeqp3` *as implemented in Netlib (left) and Intel's MKL (right). Our implementation was done in C, and was 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/


Partial Factorization Times, cols processed=1000, b=250
×10^-7
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Time [s] / n^2
In Core
SSD
HDD
0
5
10
15
n
×10^4

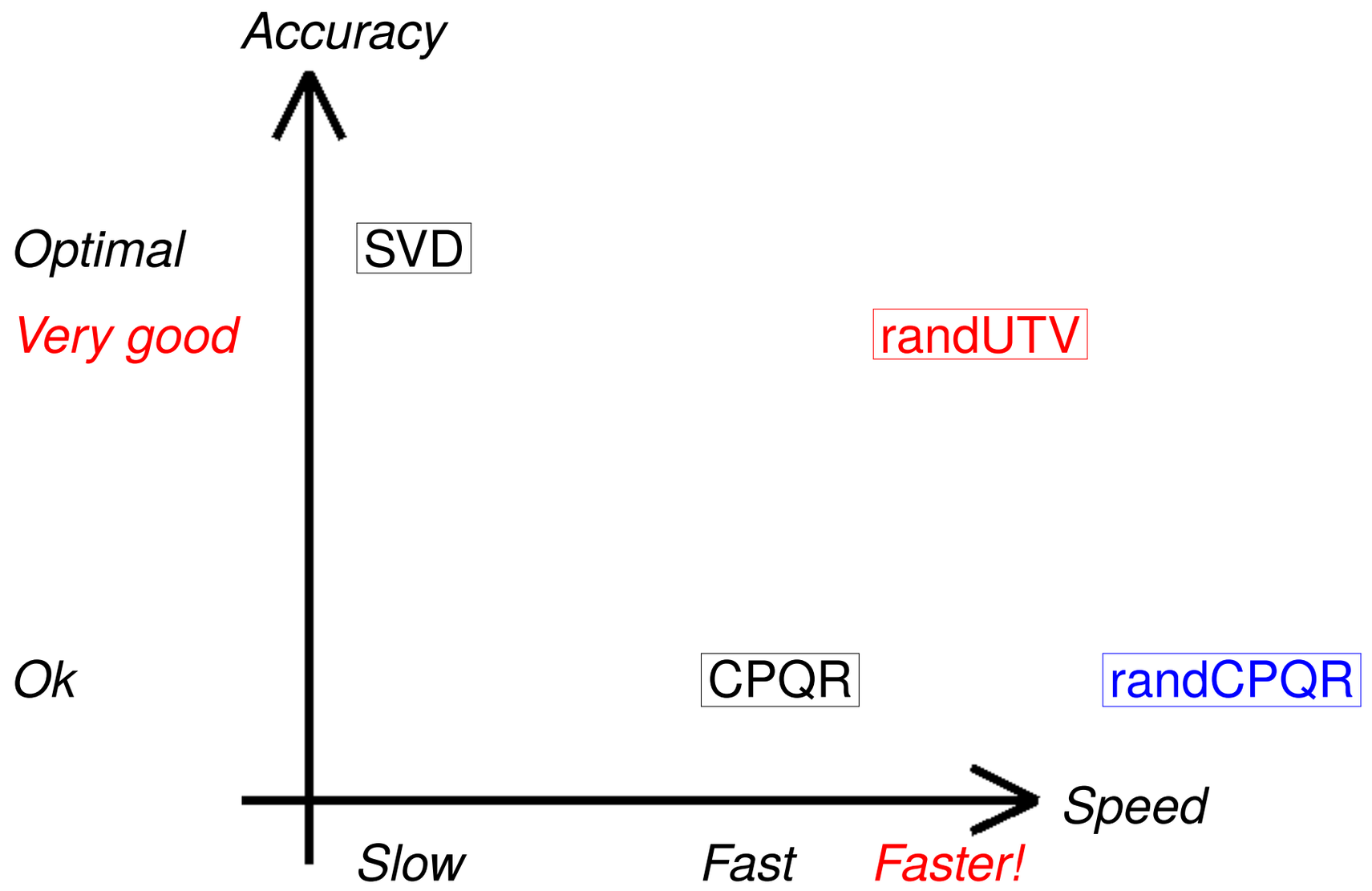
Randomized Column Pivoted QR (randCPQR)

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:



Randomized Column Pivoted QR (randCPQR)

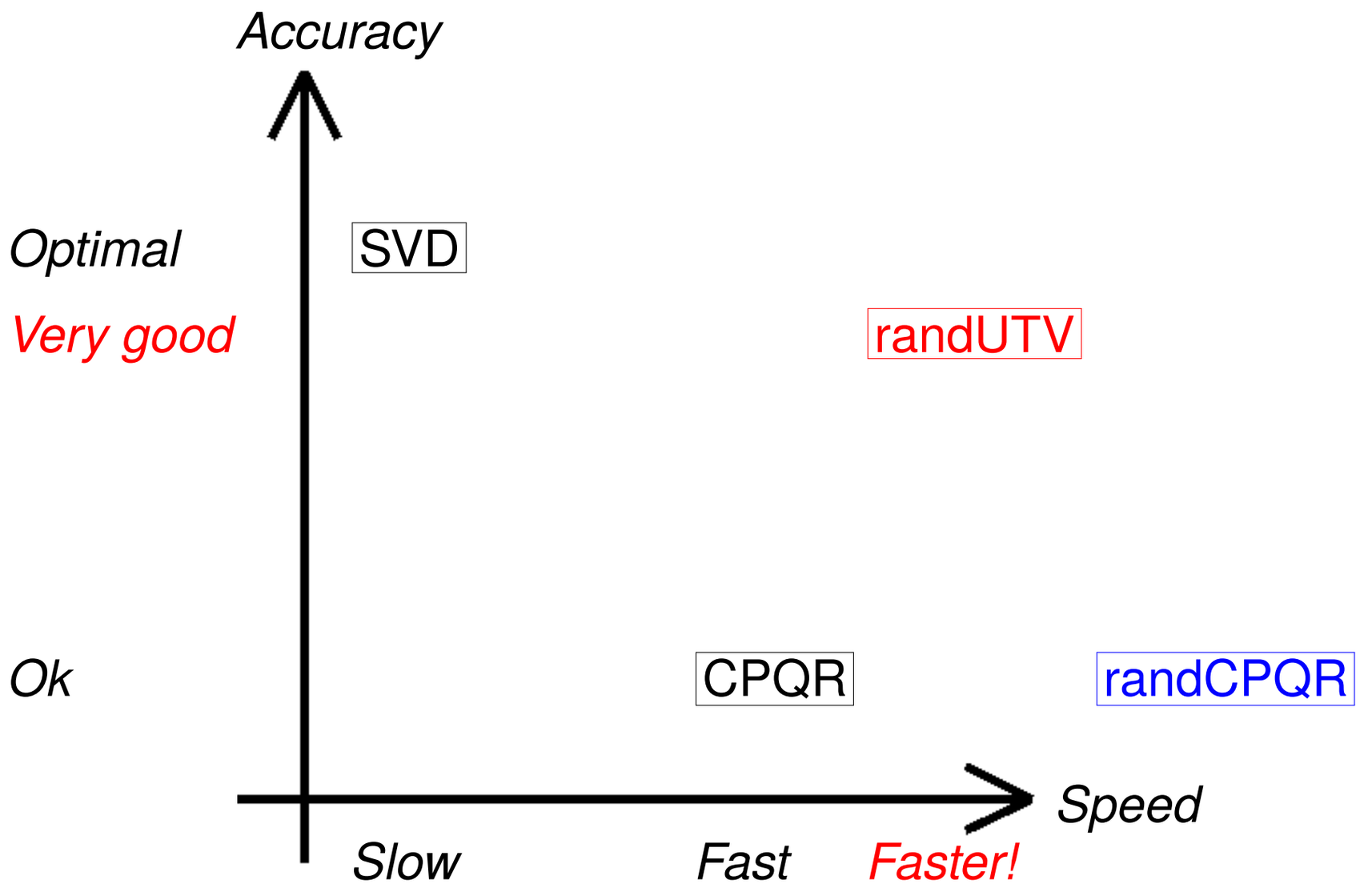
For the task of computing low-rank approximations to matrices, the classical choice is between SVD and column pivoted QR (CPQR). SVD is slow, and CPQR is inaccurate:



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).

Randomized Column Pivoted QR (randCPQR)

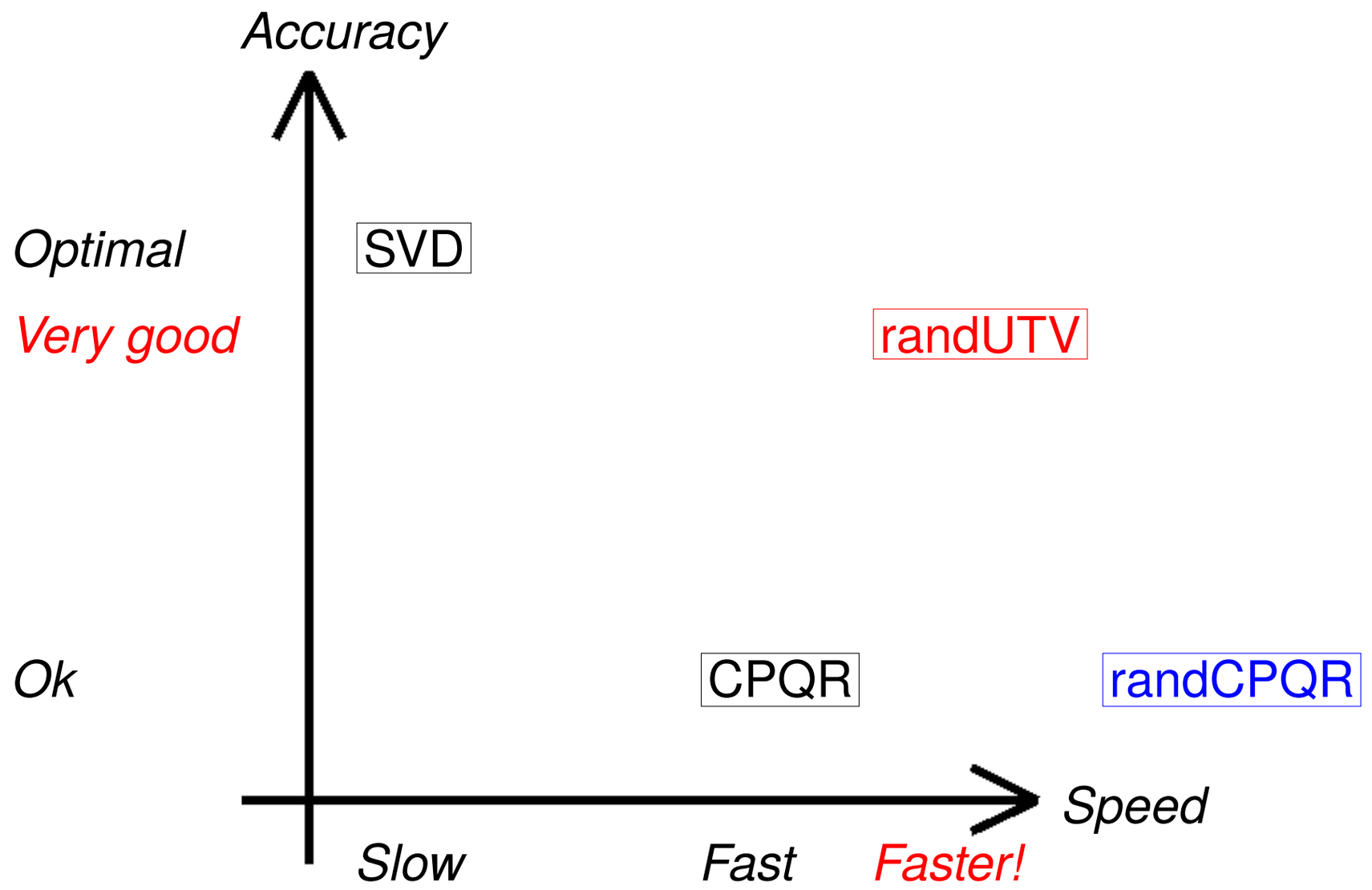
For the task of computing low-rank approximations to matrices, the classical choice is between SVD and column pivoted QR (CPQR). SVD is slow, and CPQR is inaccurate:



Future work: Continued development or randCPQR and randUTV. Adapt to different computing architectures (distributed memory, out-of-core, etc). Theory. Exploit information that is currently wasted. Multiple sweeps version. Algorithm-by-blocks.

Randomized Column Pivoted QR (randCPQR)

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:



Block Krylov methods: For partial factorizations of sparse matrices, integrate ideas from Krylov methods. Explore design space between the basic RSVD and classical "single-vector" Krylov methods. Recent work by Musco & Musco; Tropp; Gu.

Key themes:

- Randomized projections are powerful tools for dimension reduction.

  Many methods involve $\frac{1}{\epsilon^2}$-type scaling. Caveat emptor.

  Randomized projections work particularly well as one part of a two-stage method:
  - **Stage A:** Use randomized projections to develop a *sketch* of the data — "where to look".
  - **Stage B:** Return to the original data set to compute high accuracy answers.

- Randomized methods can reduce *communication* in matrix computations:

  $\rightarrow$ Key to high performance on GPUs, out-of-core, distributed memory, ...

  - randUTV and randCPQR offer much higher speed than existing methods.

Future directions:

Postdoc positions available!

- Continued work on algorithms for computing *full* factorizations of matrices.
- Accelerated solvers for $\mathbf{Ax} = \mathbf{b}$ for a "general" $\mathbf{A}$.
- Compression of rank-structured matrices ($\mathcal{H}$-matrices, HBS/HSS matrices, etc).
- Randomized block Krylov methods.
- Integration with applications in big data and scientific computing.

References:

- N. Halko, P.G. Martinsson, J. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." *SIAM Review*, **53**(2), pp. 217–288, 2011.

- P.G. Martinsson, "Randomized methods for matrix computations." Arxiv.org #1607.01649. In proceedings book for 2016 PCMI summer school.

- P.G. Martinsson, G. Quintana-Ortí, N. Heavner, and R. van de Geijn, "Householder QR Factorization With Randomization for Column Pivoting (HQRRP)." *SIAM J. on Scientific Comp.*, **39**(2), pp. C96-C115, 2017.

- P.G. Martinsson, G. Quintana-Ortí, N. Heavner, "randUTV: A blocked randomized algorithm for computing a rank-revealing UTV factorization." Accepted for publication by *ACM TOMS*. arxiv.org #1703.00998.

- J. Duersch & M. Gu, "Randomized QR with Column Pivoting", *SIAM Journal on Scientific Computing*, **39**(4), 2017.

Software for UTV: `https://github.com/flame/randutv`

Software for CPQR: `https://github.com/flame/hqrrp`