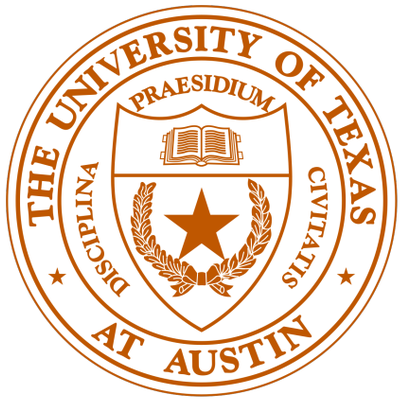
Randomized Algorithms for Computing Full Matrix Factorizations

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:



Background: Randomized methods such as the "randomized SVD (RSVD)" have proven effective at computing low rank approximations to matrices.

Topic of this talk: How to use methods based on randomized projections to efficiently compute *full* factorizations. Say you want *all* of the eigenvalues, or that the effective rank is not that much smaller than the matrix dimensions (say 10% or 50%).

Themes:

- Use randomization to reduce *communication* rather than flops.
- Quest for algorithms that spend most flops in matrix-matrix multiplications.
- Communication constrained environments:
  - Distributed memory computing.
  - Matrices stored "out-of-core" (say on a hard drive or an SSD).
  - GPU computing.

Outline:

- Review of randomized SVD.
- A randomized method for computing a UTV decomposition.
  (The UTV decomposition is a relaxation of a singular value decomposition.)
- A randomized method for computing a column pivoted QR decomposition.

Background/review: Randomized singular value decomposition (RSVD)

Problem: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, where $k \ll \min(m, n)$, we seek to compute an approximate partial singular value decomposition:

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

with $\mathbf{U}$ and $\mathbf{V}$ having orthonormal columns, and $\mathbf{D}$ diagonal.

Solution: Pick an over-sampling parameter p, say $p = 5$. Then proceed as follows:

1. Draw an $n \times (k + p)$ Gaussian random matrix $\mathbf{R}$. `R = randn(n,k+p)`
2. Form the $m \times (k + p)$ sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{R}$. `Y = A * R`
3. Form an $m \times (k + p)$ orthonormal matrix $\mathbf{Q}$ s. t. $\text{col}(\mathbf{Y}) = \text{col}(\mathbf{Q})$. `[Q, ~] = qr(Y)`
4. Form the $(k + p) \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\,\mathbf{A}$. `B = Q' * A`
5. Compute the SVD of $\mathbf{B}$ (small!): $\mathbf{B} = \hat{\mathbf{U}}\,\mathbf{D}\,\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\,\hat{\mathbf{U}}$. `U = Q * Uhat`
7. Optional: Truncate the last p terms in the computed factors.

Background/review: Randomized singular value decomposition (RSVD)

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

Background/review: Randomized singular value decomposition (RSVD)

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 **R**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AR}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

- Often faster than alternative algorithms for low-rank approximation (CPQR, Krylov, ...) on traditional CPU based platforms.
- Order of magnitude acceleration for data stored *out-of-core.*
- Single pass algorithms have been developed for *streaming environments.*
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn\log k)$.
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Well understood mathematically — detailed performance analysis.

Question: Can these ideas be applied for a *full* rank factorization?

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

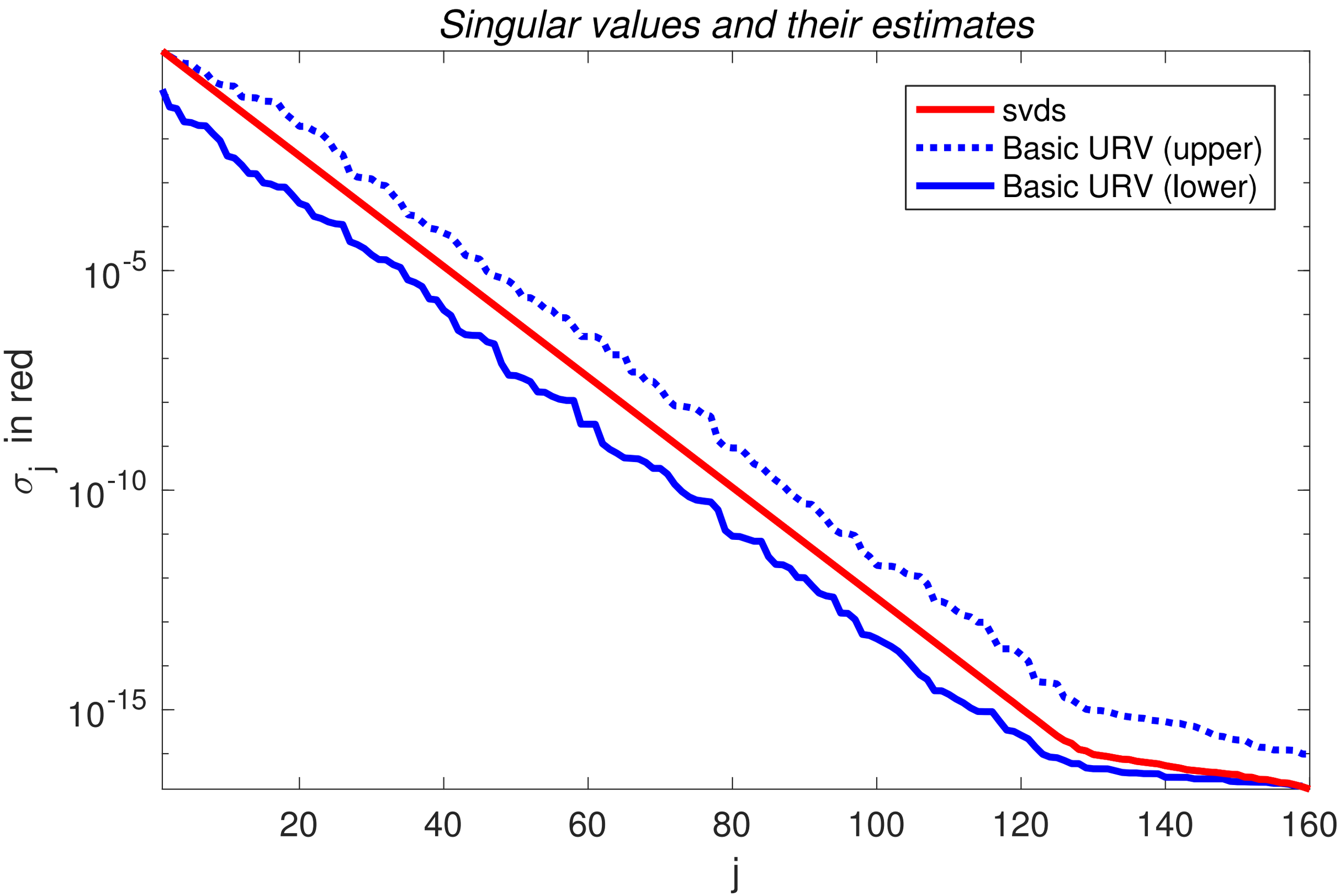
$$\nu_k = \sigma_k(\mathbf{T}(1:k, 1:k)),$$
$$\tau_{k+1} = \sigma_1(\mathbf{T}((k+1):n, (k+1):n)),$$

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

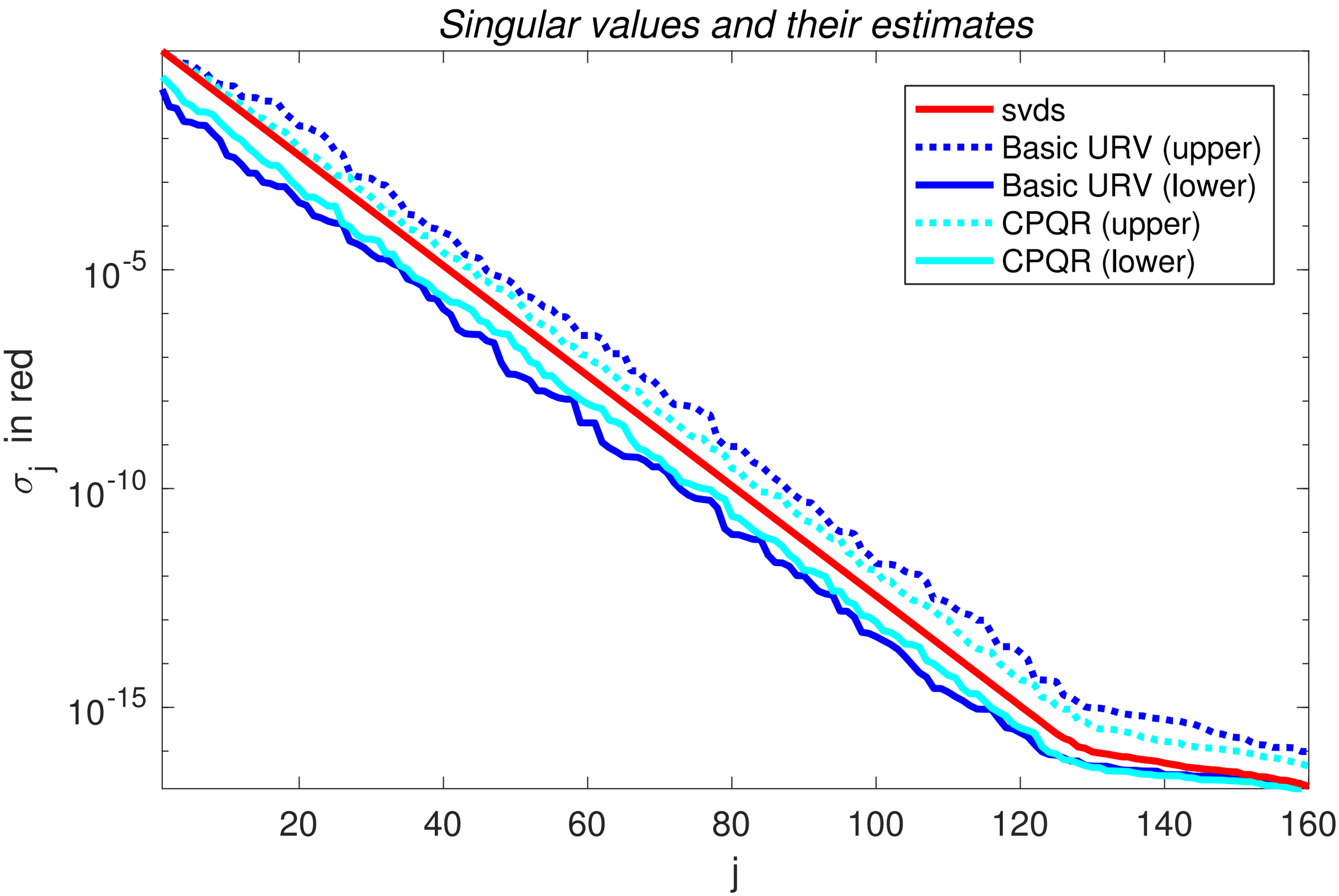
$$\nu_k \leq \sigma_k(\mathbf{A}) \leq \tau_k.$$

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

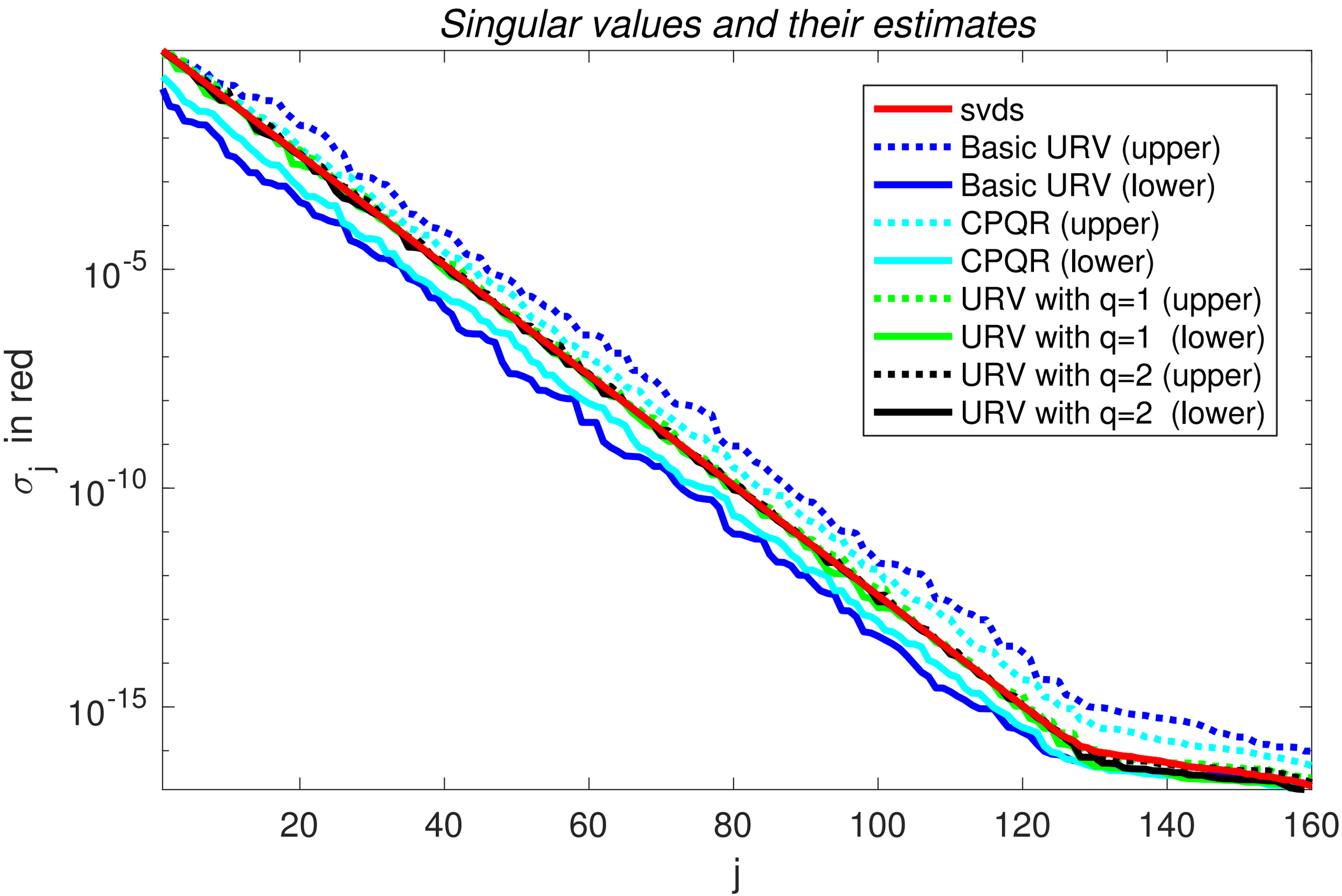
Numerical experiments illustrating the errors in the URV factorization



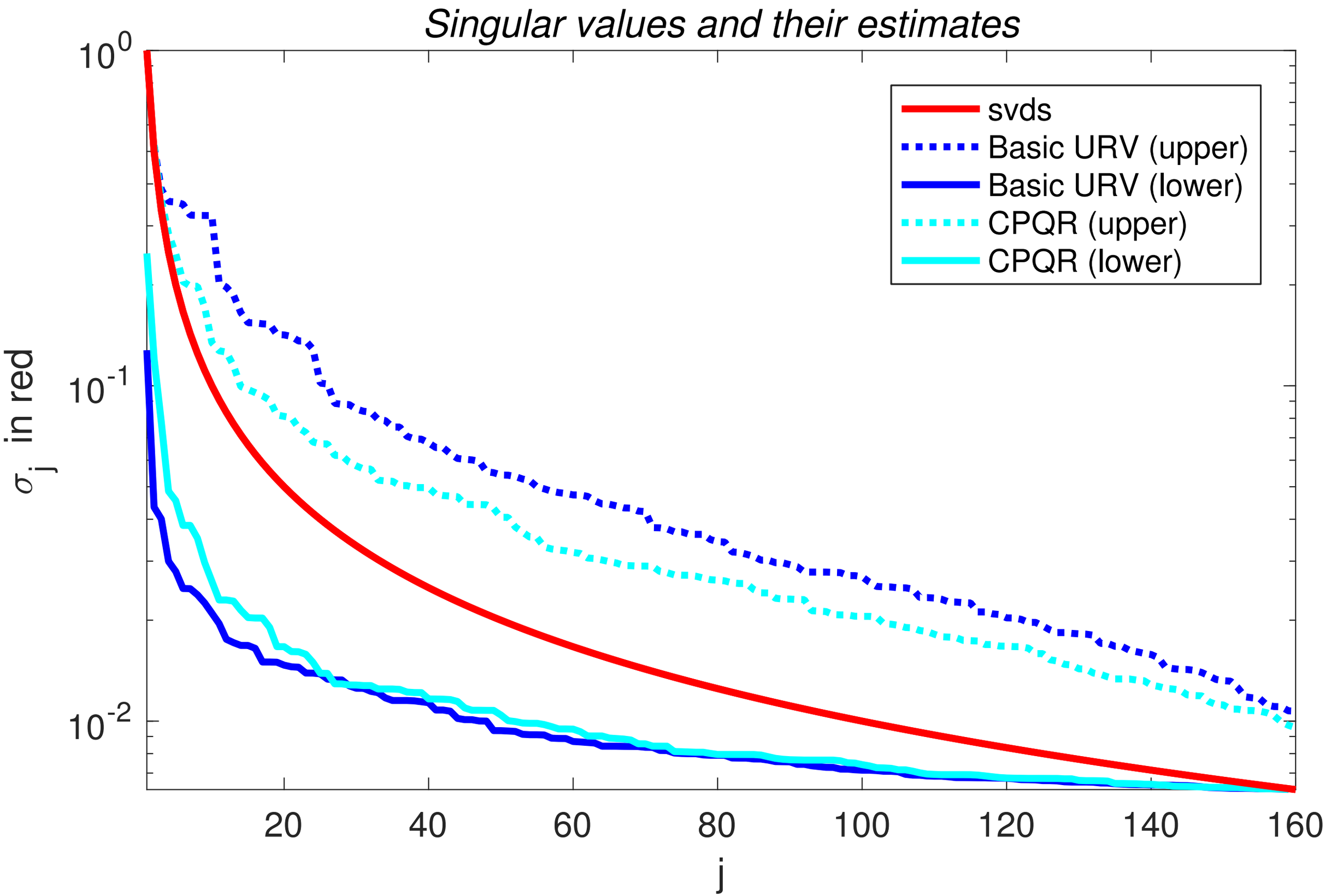
Numerical experiments illustrating the errors in the URV factorization



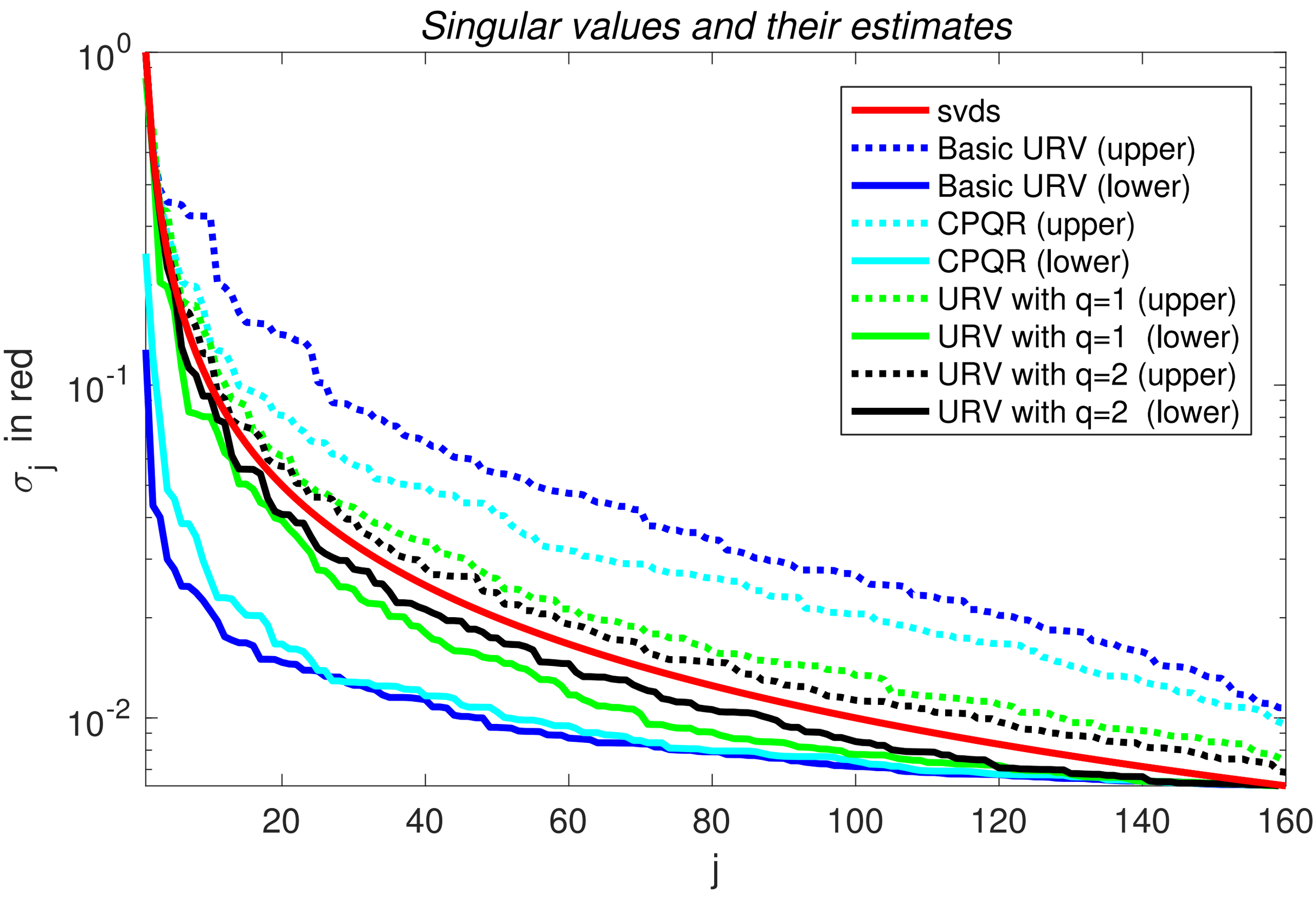
Numerical experiments illustrating the errors in the URV factorization



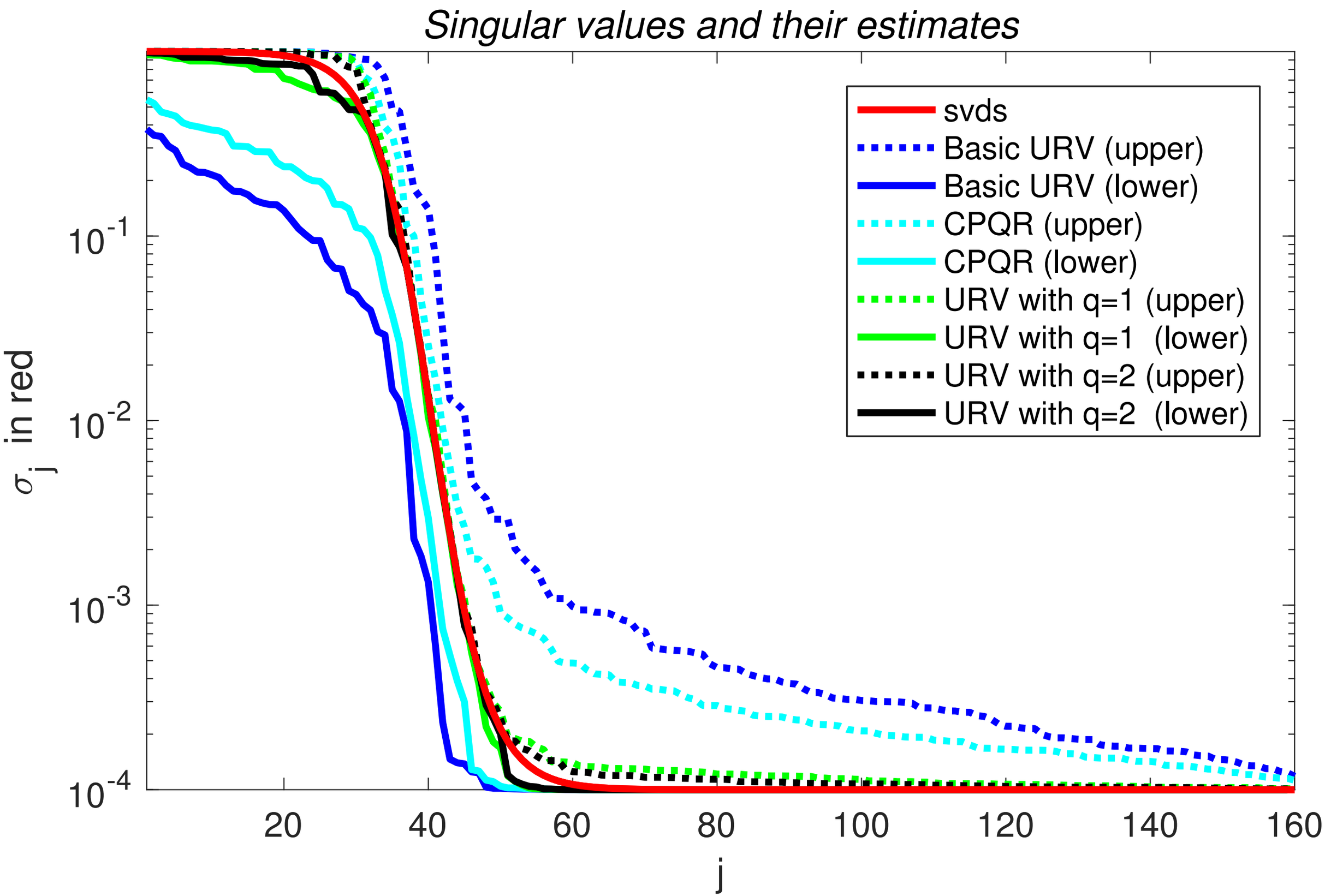
Numerical experiments illustrating the errors in the URV factorization



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

$$\begin{array}{cccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array} \tag{1}$$

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 $\mathbf{A}$.

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

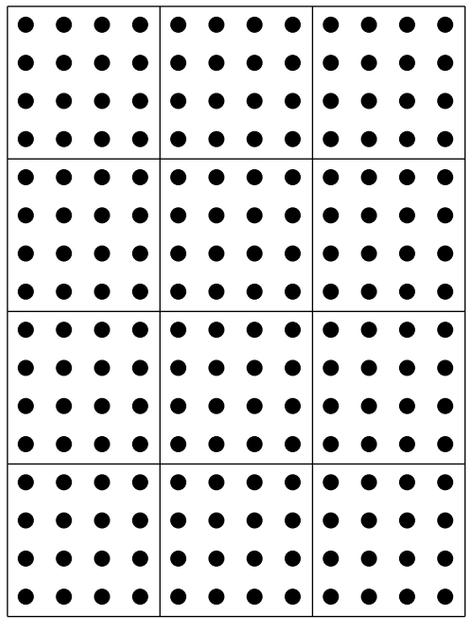
The UTV decomposition: Overview of proposed algorithm randUTV

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:



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

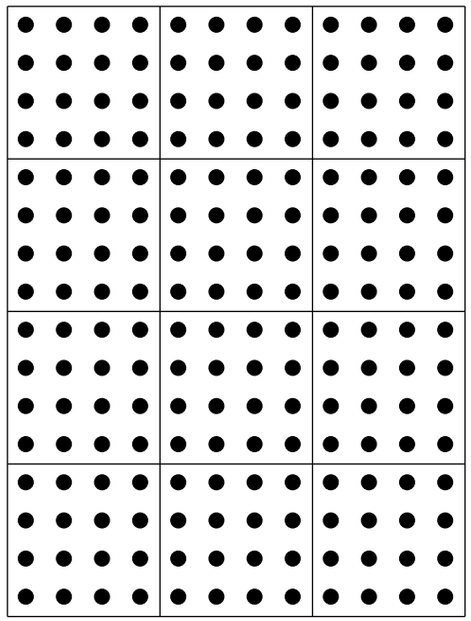
The UTV decomposition: Overview of proposed algorithm randUTV

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

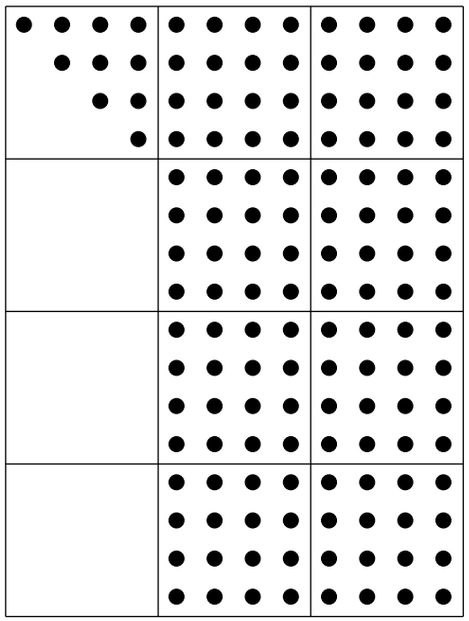
$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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:



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



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

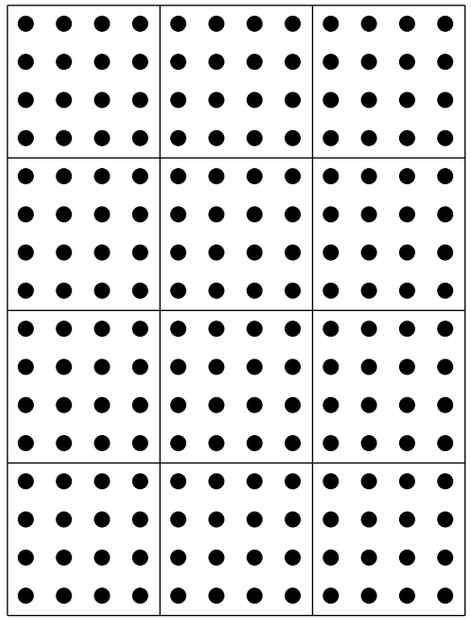
The UTV decomposition: Overview of proposed algorithm randUTV

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

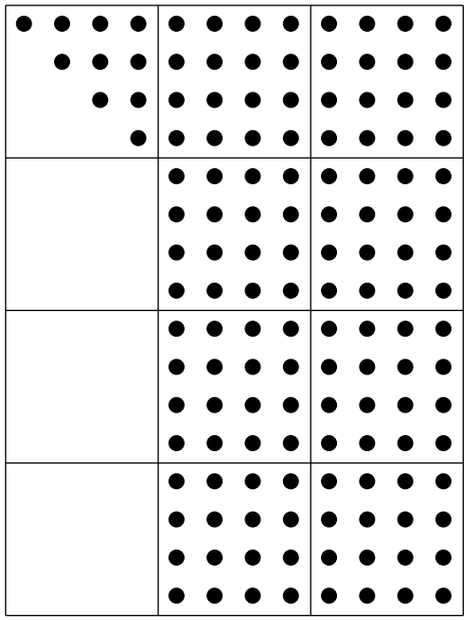
$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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

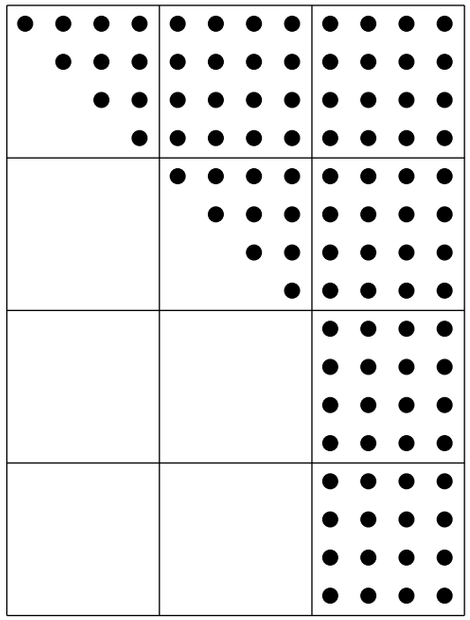
The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



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



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



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

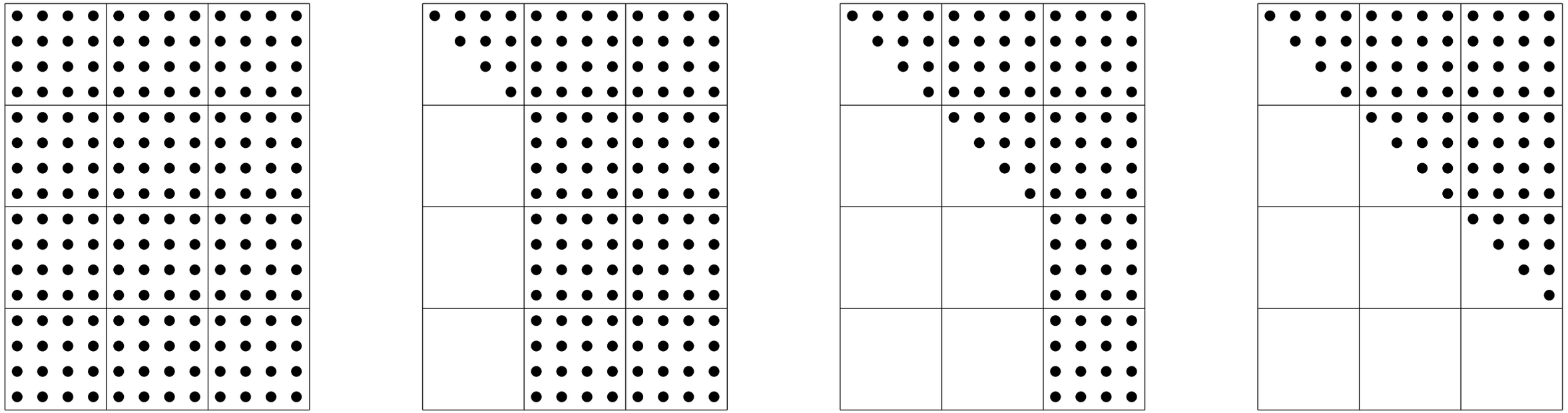
The UTV decomposition: Overview of proposed algorithm randUTV

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:

$$\mathbf{Y} = (\mathbf{A}^*\mathbf{A})^q \mathbf{A}^*\mathbf{G}, \qquad [\mathbf{V}, \sim] = \mathtt{qr}(\mathbf{Y}),$$

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

(Over-sampling can be used as well.)

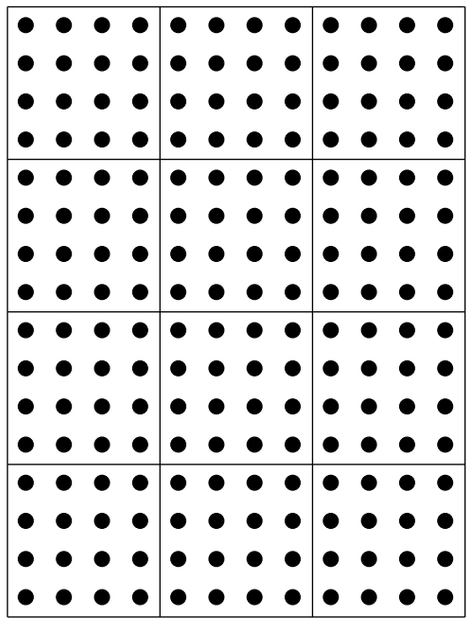
The UTV decomposition: Overview of proposed algorithm randUTV

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:



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

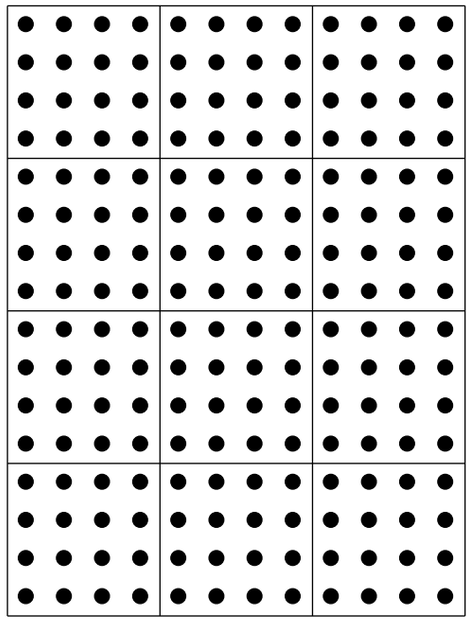
The UTV decomposition: Overview of proposed algorithm randUTV

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

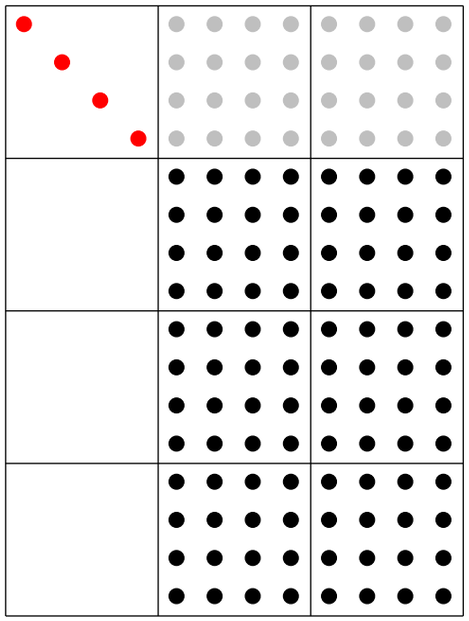
$$\underset{m \times n}{\mathbf{A}} = \underset{m \times n}{\mathbf{U}} \quad \underset{n \times n}{\mathbf{T}} \quad \underset{n \times n}{\mathbf{V}^*},$$

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:



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



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

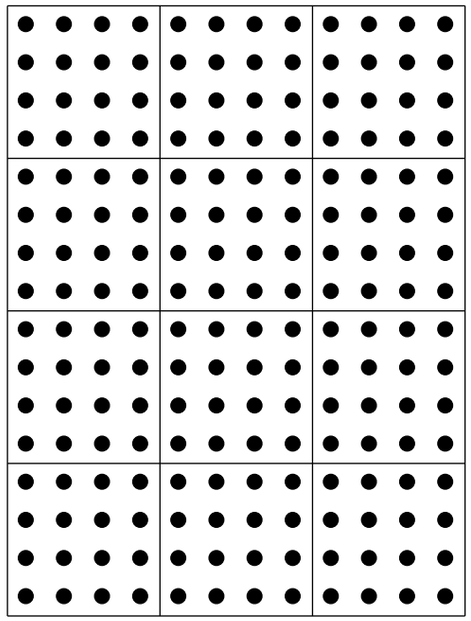
The UTV decomposition: Overview of proposed algorithm randUTV

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

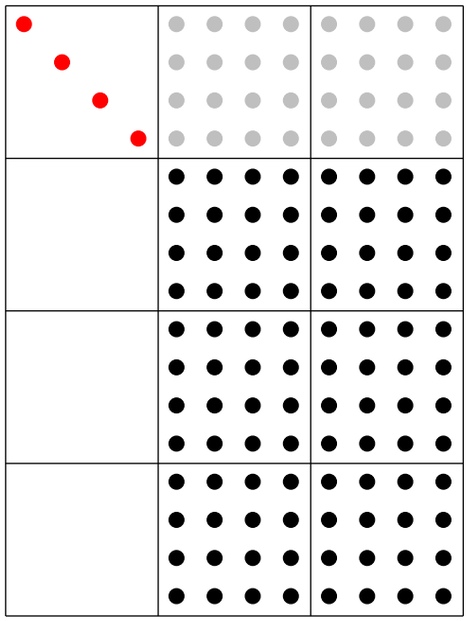
$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{T} & \mathbf{V}^*, \\ m \times n & & m \times n & n \times n & n \times n \end{array}$$

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

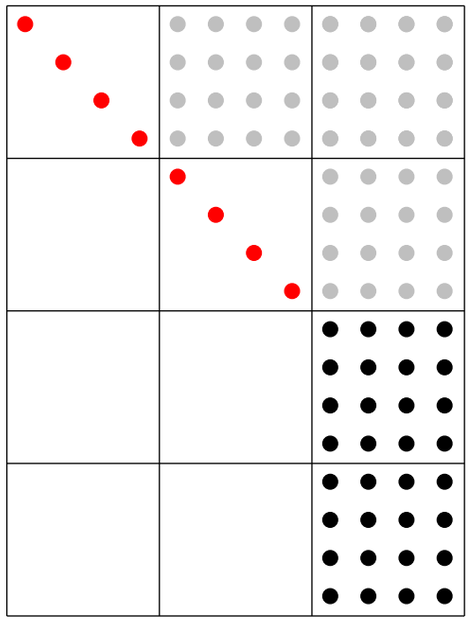
The technique proposed drives $\mathbf{A}$ to upper triangular form via unitary transformations:



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



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



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

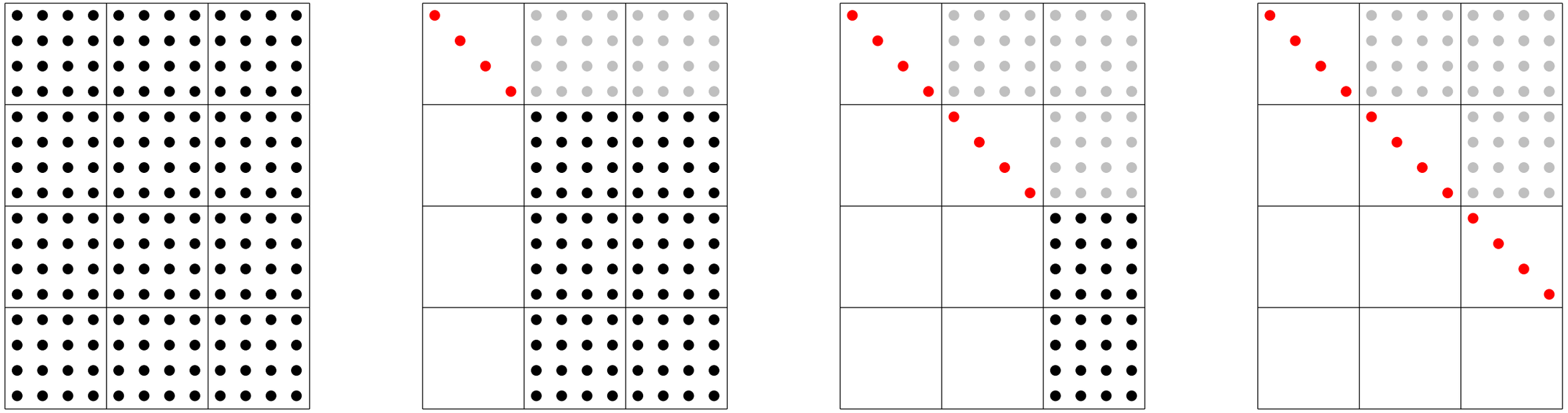
The UTV decomposition: Overview of proposed algorithm randUTV

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.

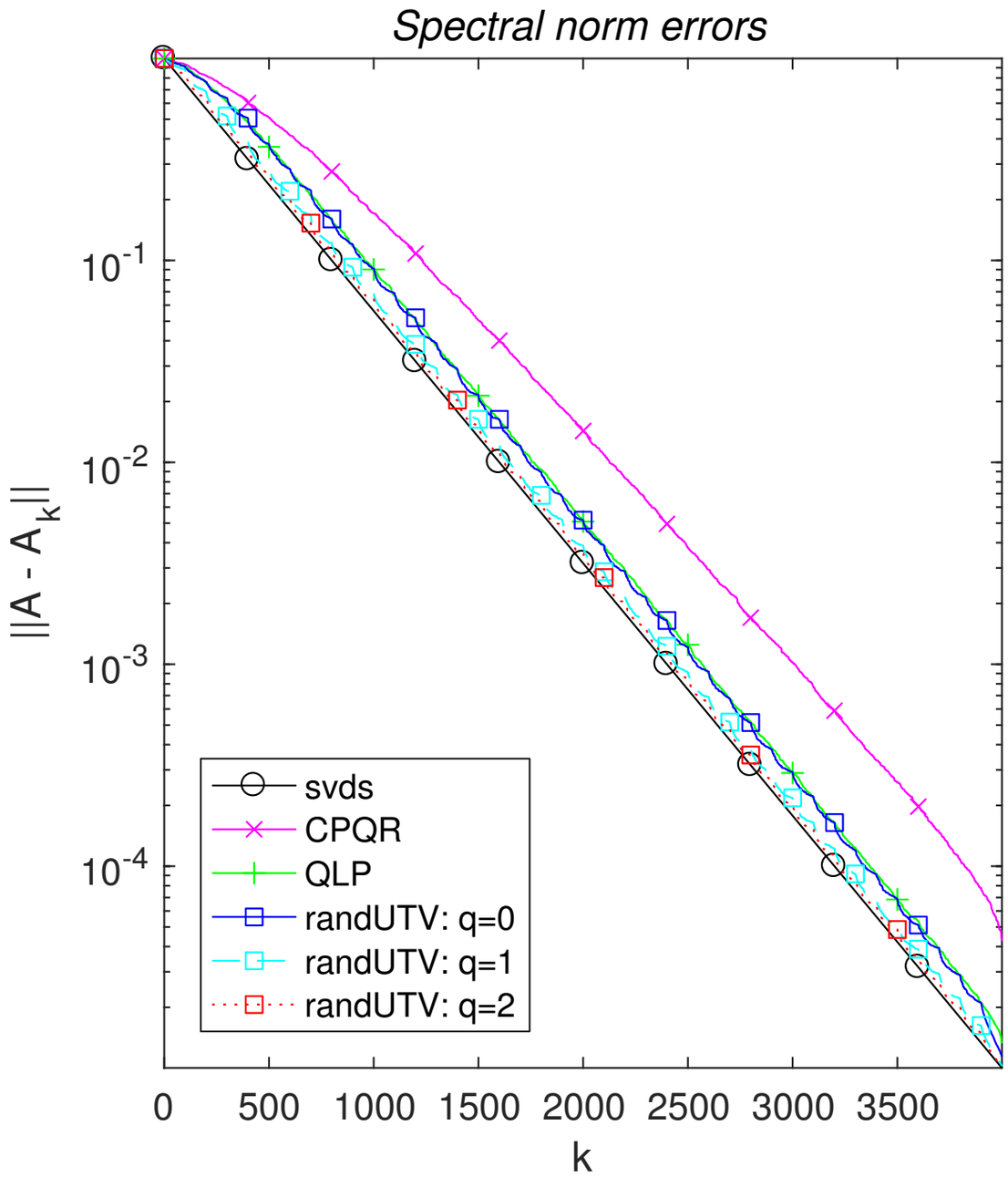
The super-diagonal elements are very small — often of relative size 10^{-5} *or so!*

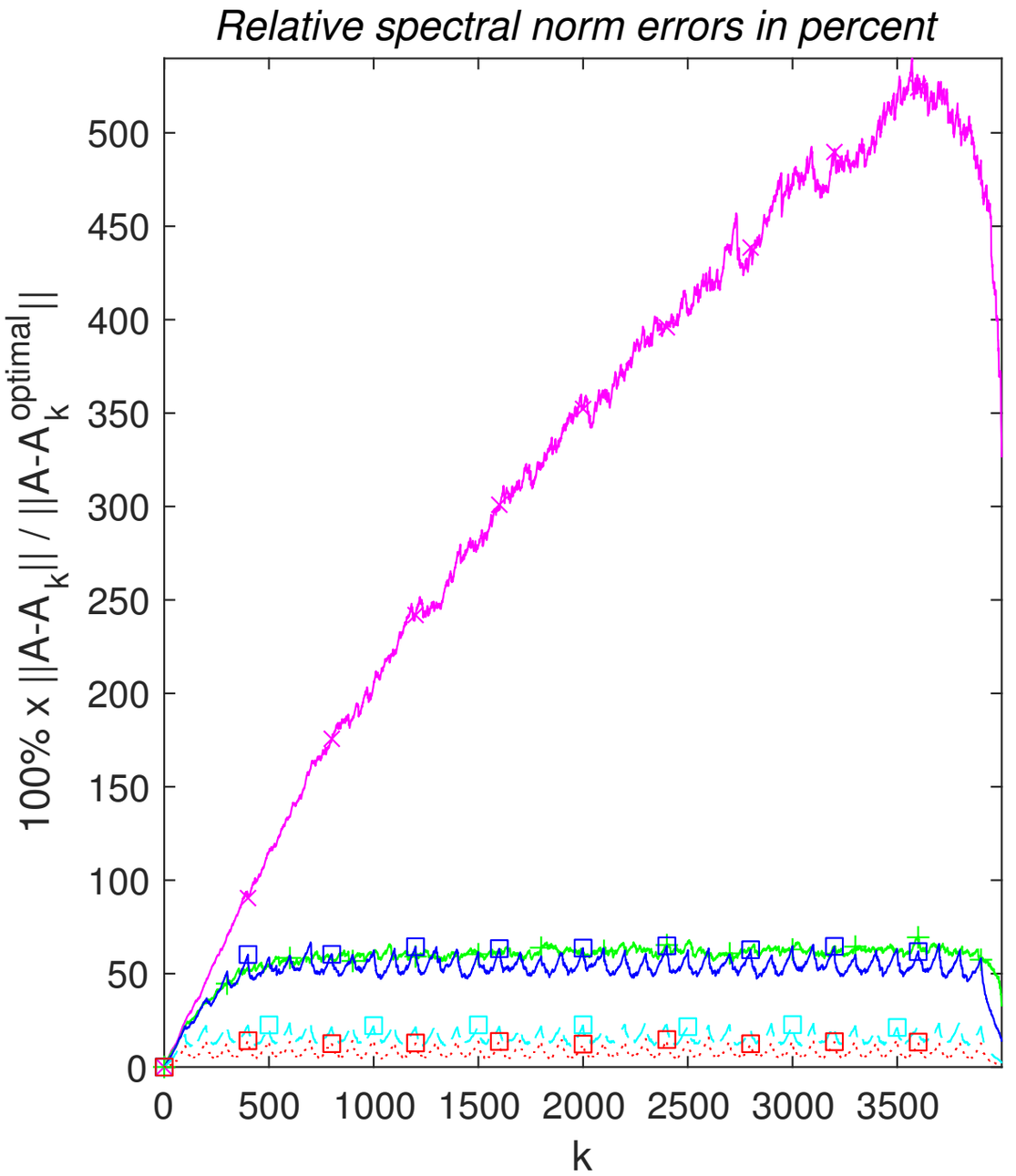
```
function [U,T,V] = randUTV(A,b,q)
  T = A;
  U = eye(size(A,1));
  V = eye(size(A,2));
  for i = 1:ceil(size(A,2)/b)
    I1 = 1:(b*(i-1));
    I2 = (b*(i-1)+1):size(A,1);
    J2 = (b*(i-1)+1):size(A,2);
    if (length(J2) > b)
      [UU,TT,VV] = stepUTV(T(I2,J2),b,q);
    else
      [UU,TT,VV] =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

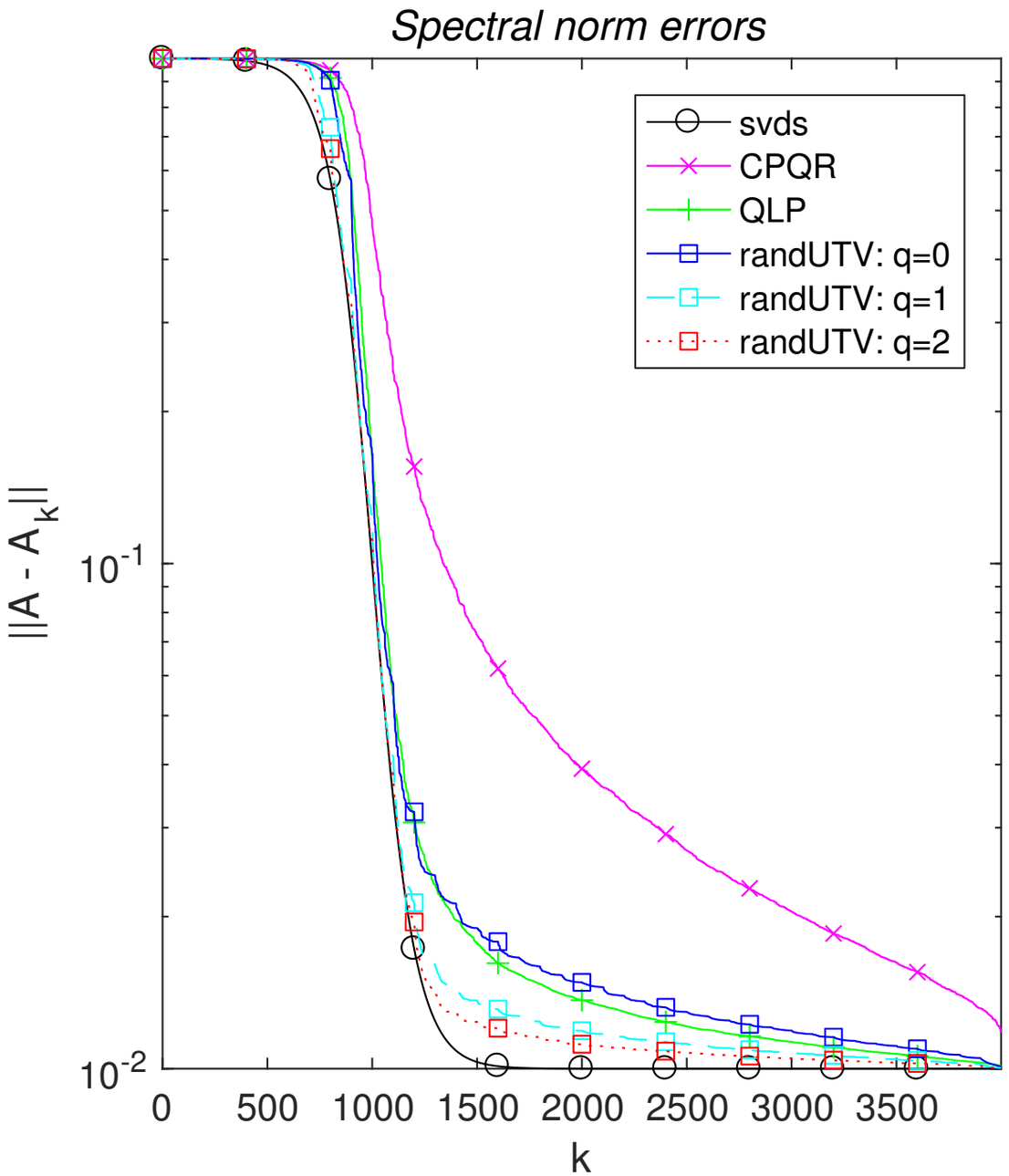
Numerical experiments illustrating the errors in the UTV factorization

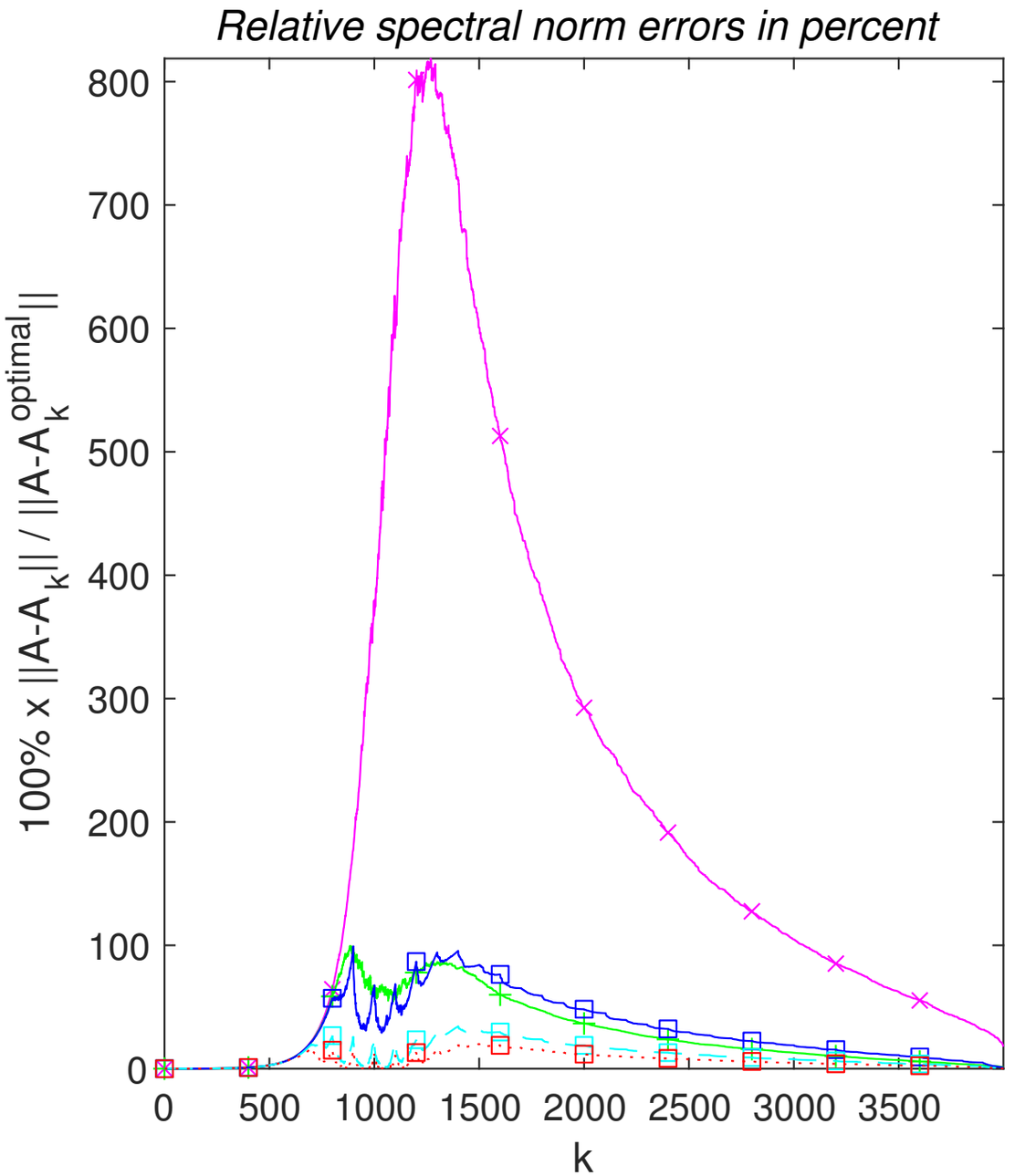




Rank-k approximation errors for the matrix "Fast Decay" of size 4000×4000. *The block size was* $b = 100$. *Left: Absolute errors in spectral norm. The black line (circles) marks the theoretically minimal errors. Right: Relative errors* $e_k^{\text{relative}} = 100\% \times \dfrac{\|\mathbf{A} - \mathbf{A}_k\|}{\|\mathbf{A}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

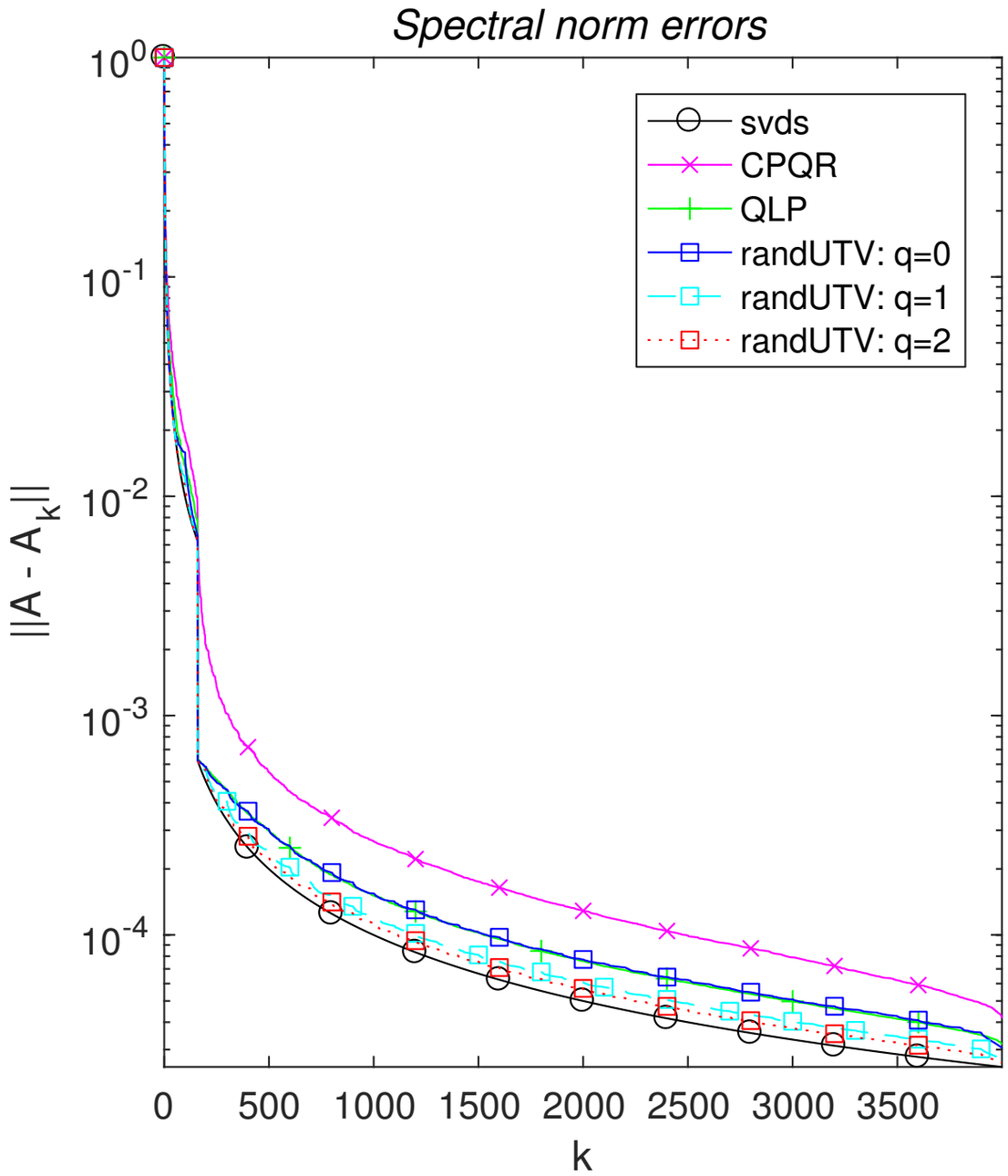
Numerical experiments illustrating the errors in the UTV factorization

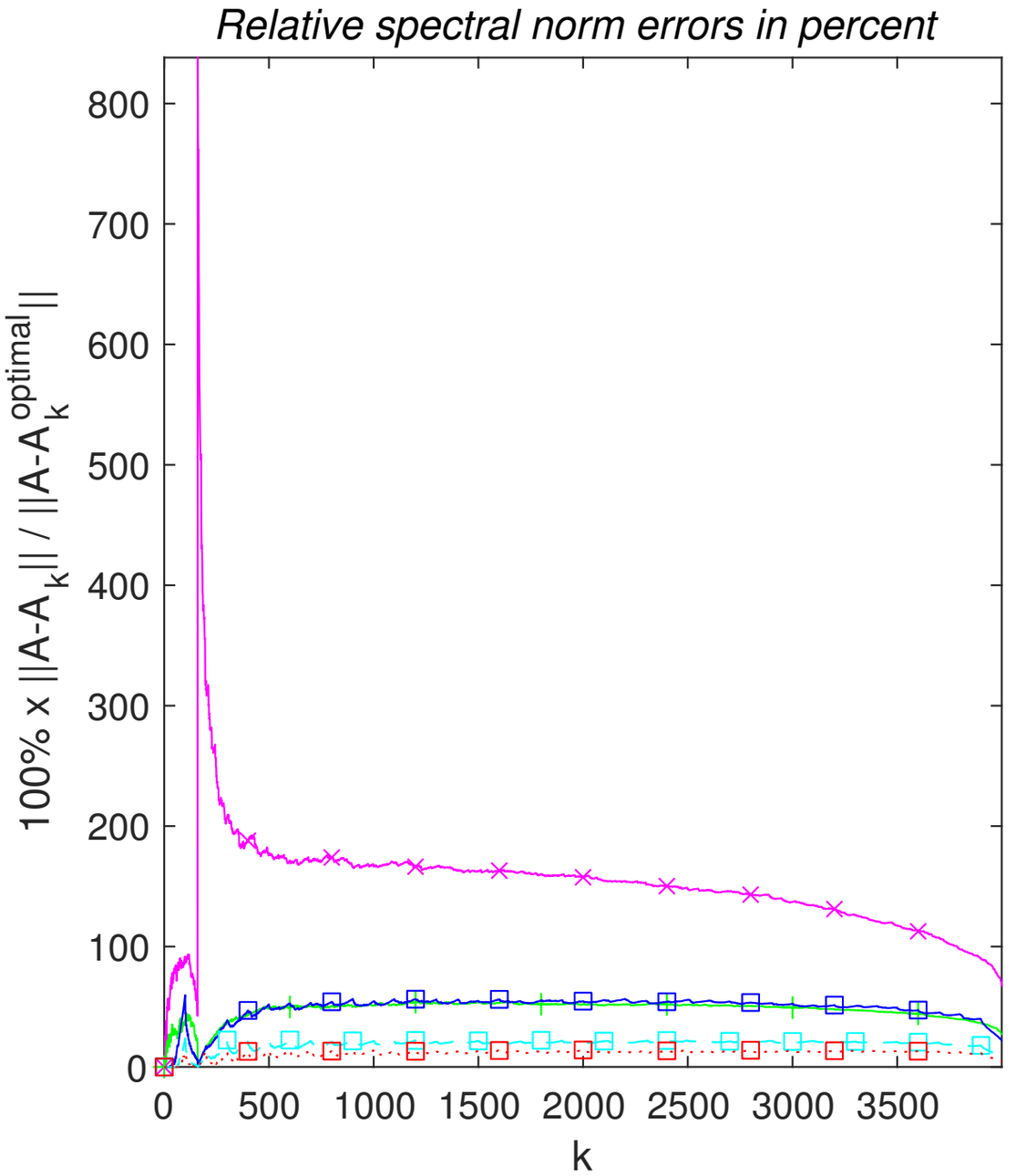




Rank-k approximation errors for the matrix "S-shape" of size 4000×4000. *The block size was* $b = 100$. *Left: Absolute errors in spectral norm. The black line (circles) marks the theoretically minimal errors. Right: Relative errors* $e_k^{\text{relative}} = 100\% \times \dfrac{\|\mathbf{A} - \mathbf{A}_k\|}{\|\mathbf{A}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

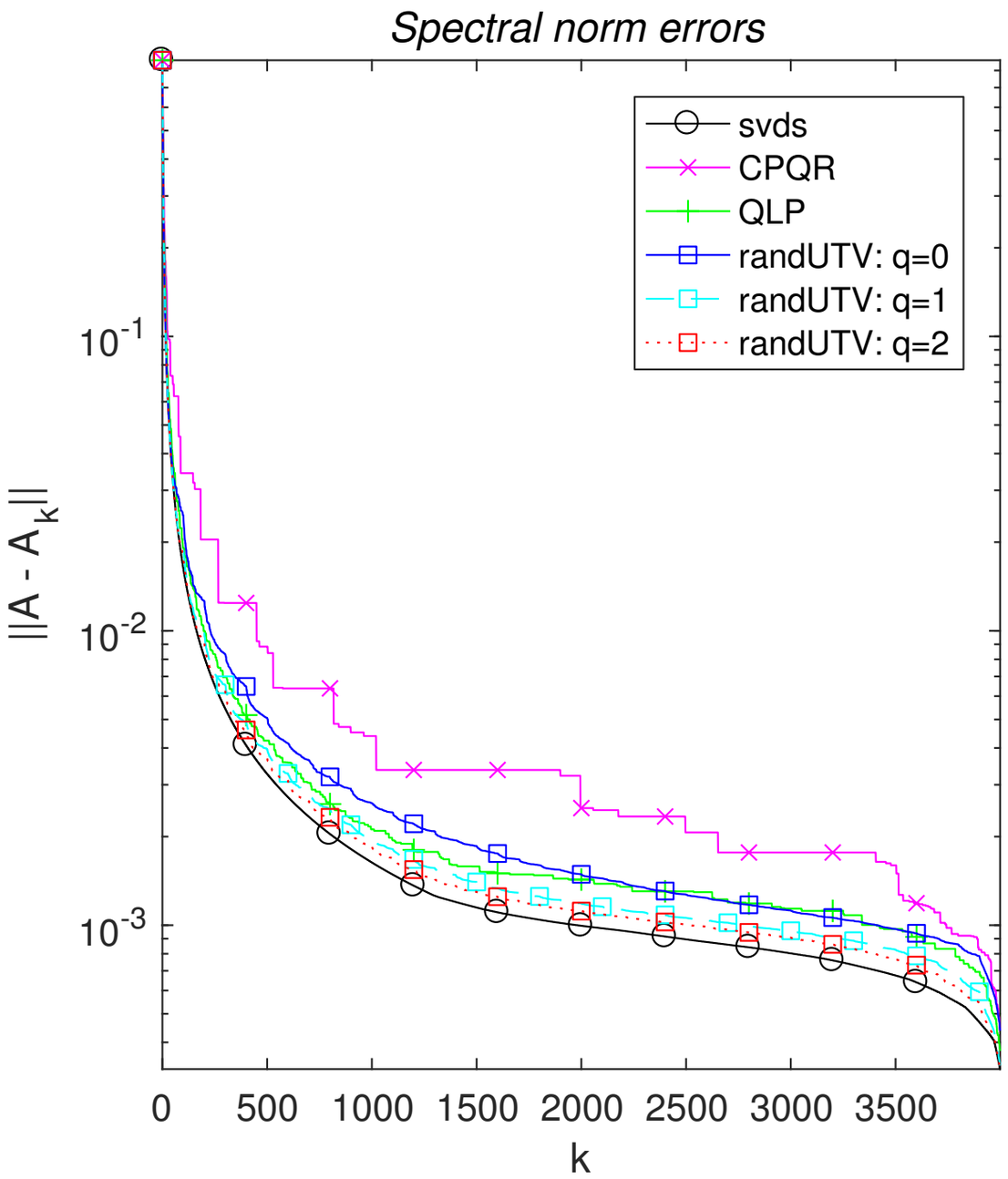
Numerical experiments illustrating the errors in the UTV factorization

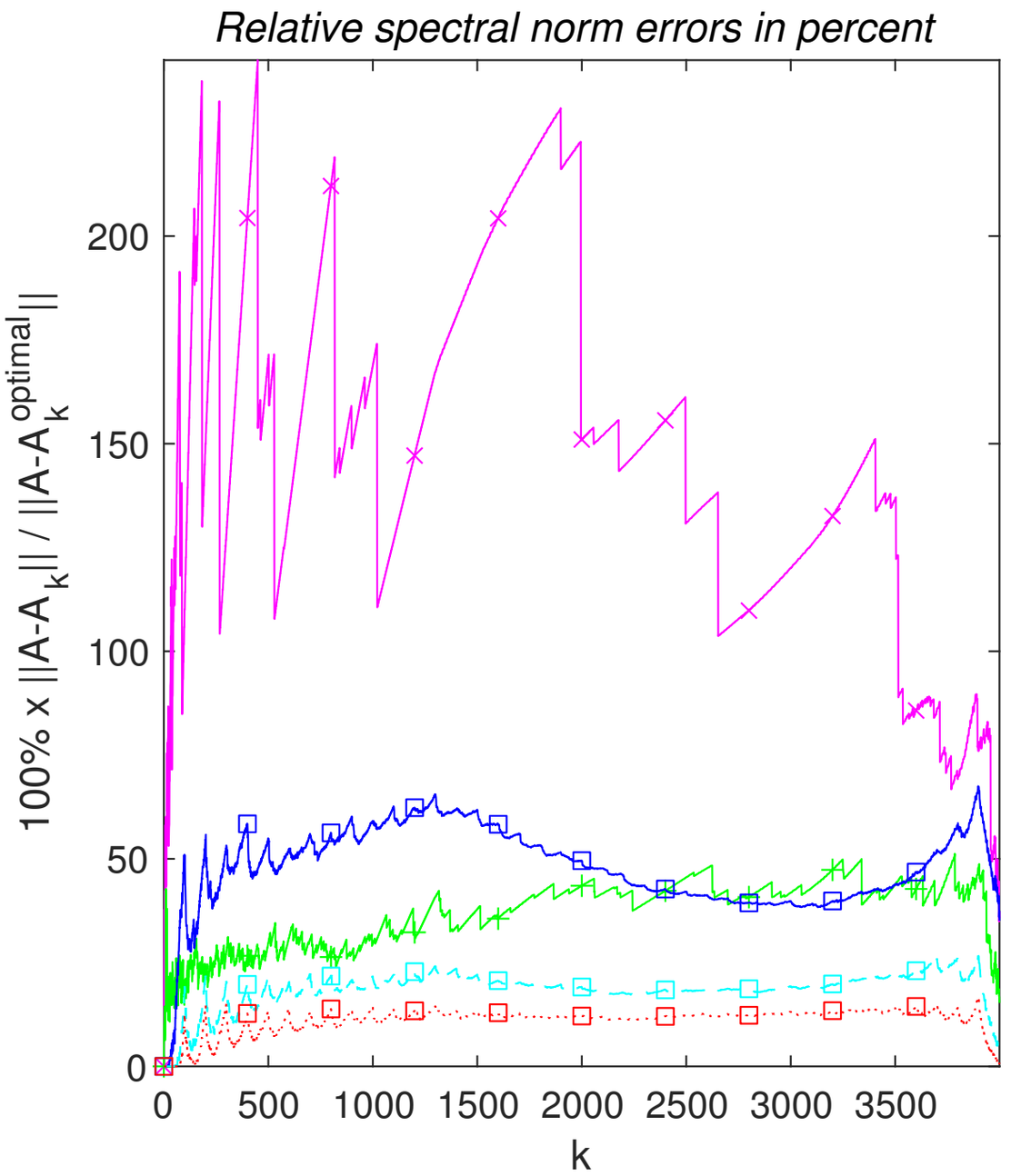




Rank-k approximation errors for the matrix "Gap" of size 4000×4000. *The block size was* $b = 100$. *Left: Absolute errors in spectral norm. The black line (circles) marks the theoretically minimal errors. Right: Relative errors* $e_k^{\text{relative}} = 100\% \times \dfrac{\|\mathbf{A} - \mathbf{A}_k\|}{\|\mathbf{A}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

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**:

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

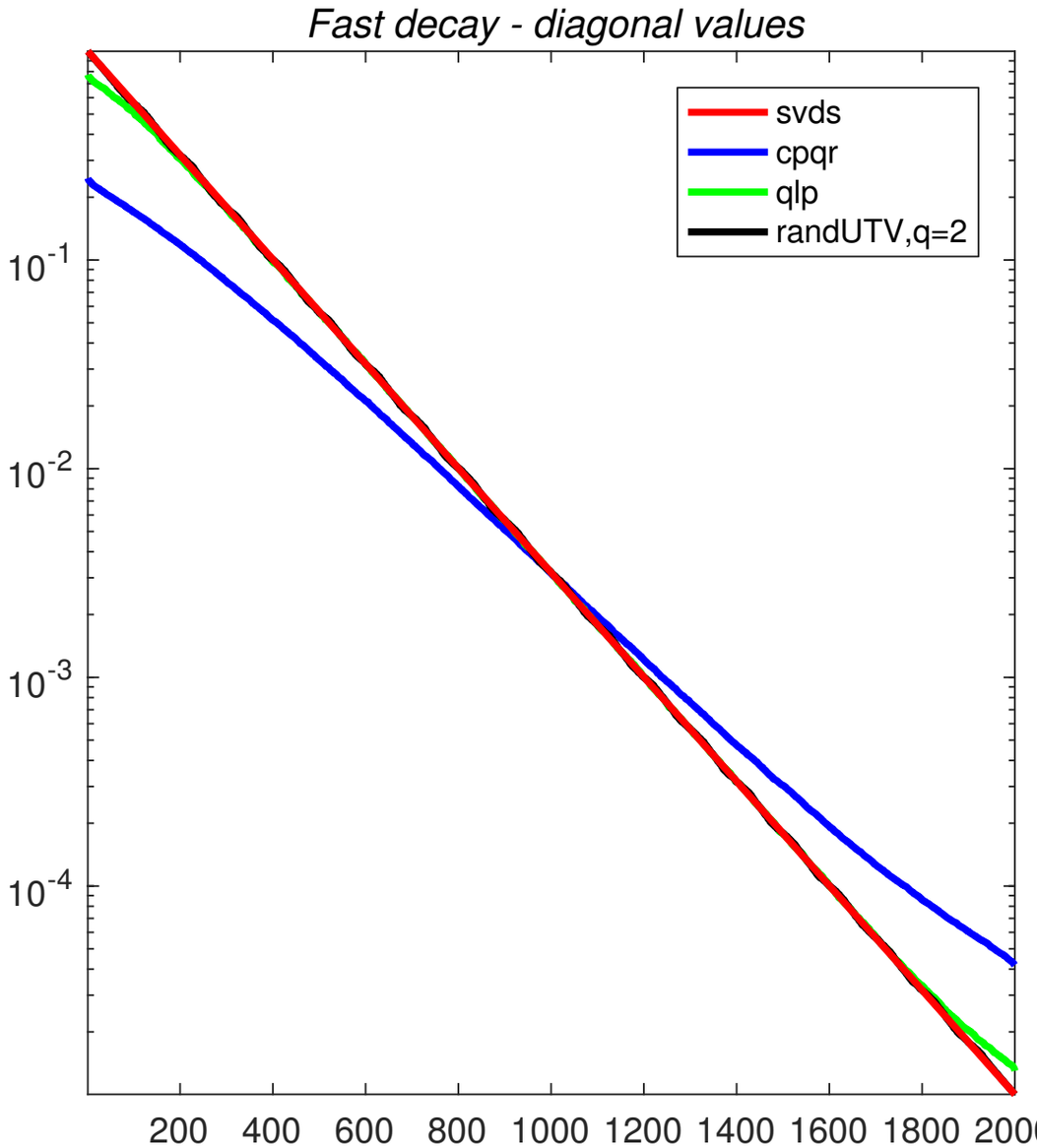
Question: How good?

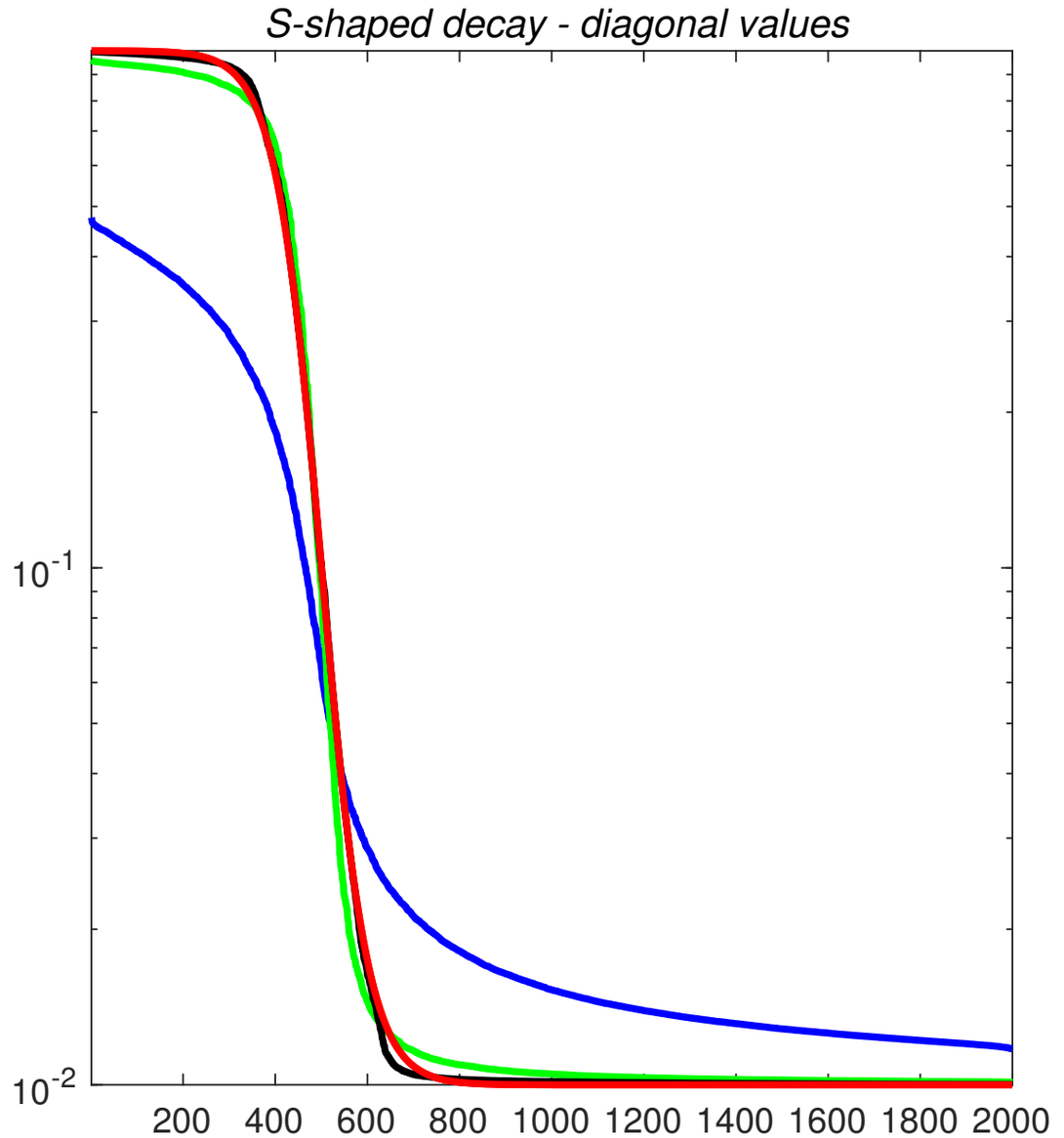
Numerical experiments illustrating how close the UTV is to the SVD

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?



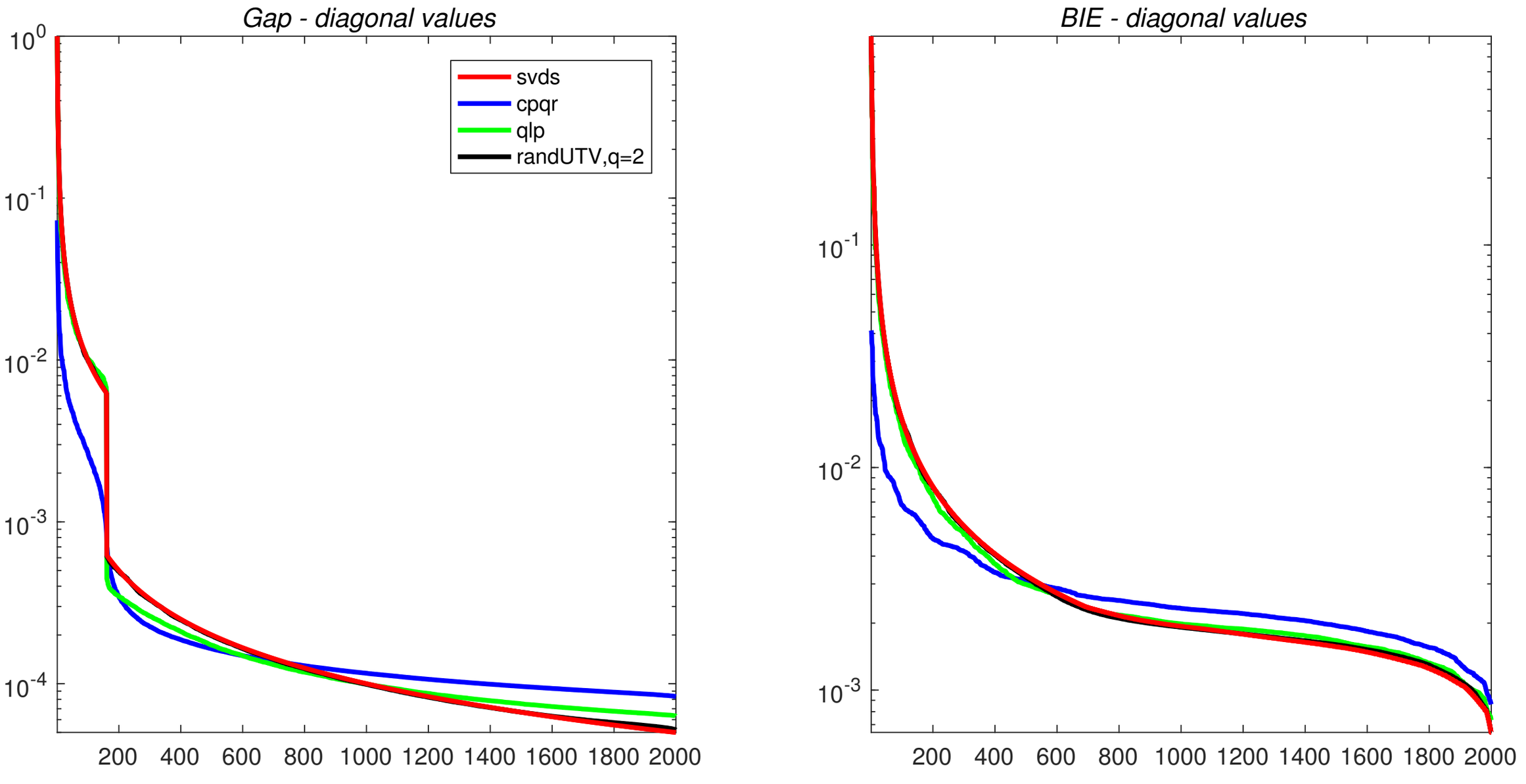


Numerical experiments illustrating how close the UTV is to the SVD

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)

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

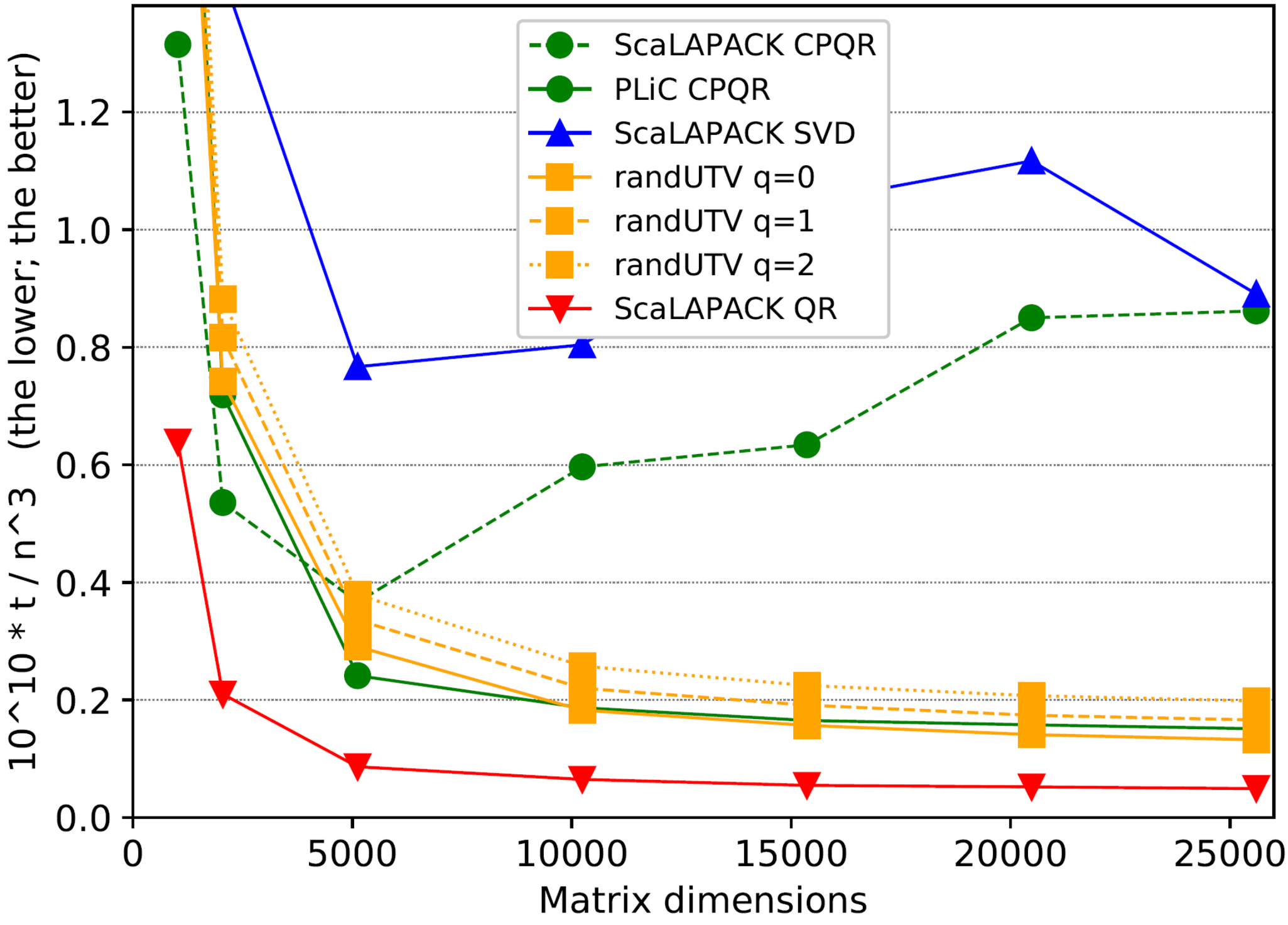
2
1.5
1
0.5
0

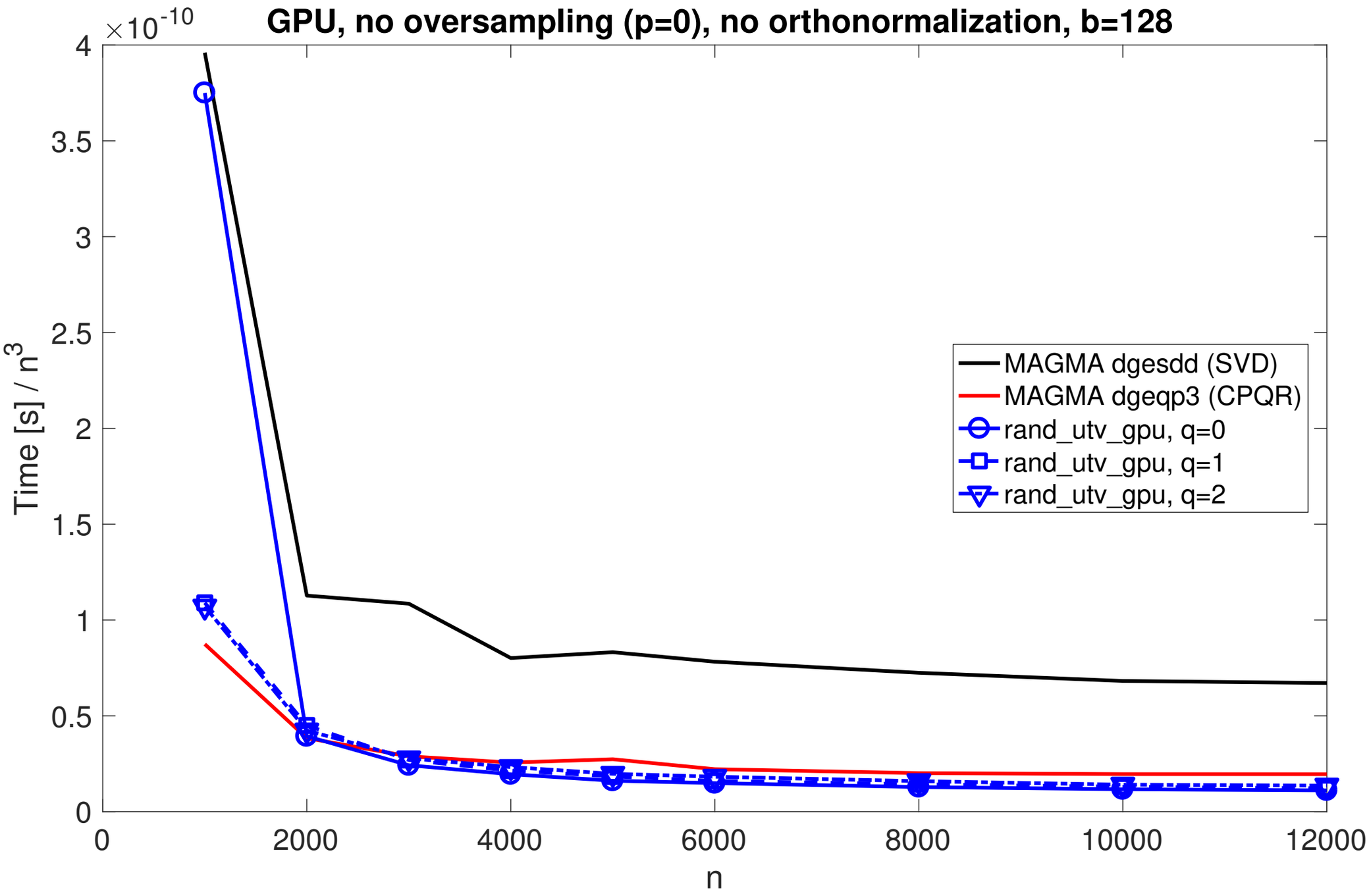
0
20480
40960
61440
81920
102400

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.

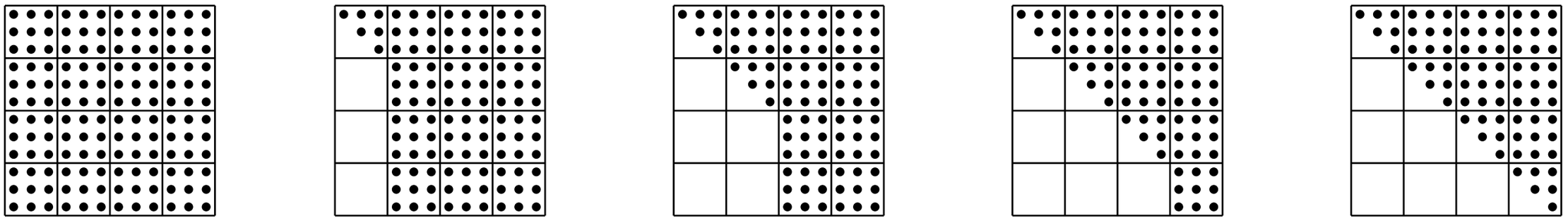
Randomized Column Pivoted QR (randCPQR)

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:

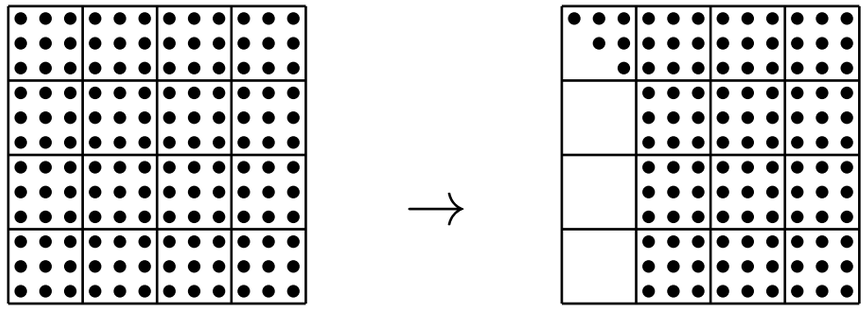


$$\mathbf{A}_0 = \mathbf{A} \qquad \mathbf{A}_1 = \mathbf{Q}_1^* \mathbf{A}_0 \mathbf{P}_1 \qquad \mathbf{A}_2 = \mathbf{Q}_2^* \mathbf{A}_1 \mathbf{P}_2 \qquad \mathbf{A}_3 = \mathbf{Q}_3^* \mathbf{A}_2 \mathbf{P}_3 \qquad \mathbf{A}_4 = \mathbf{Q}_4^* \mathbf{A}_3 \mathbf{P}_4$$

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

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

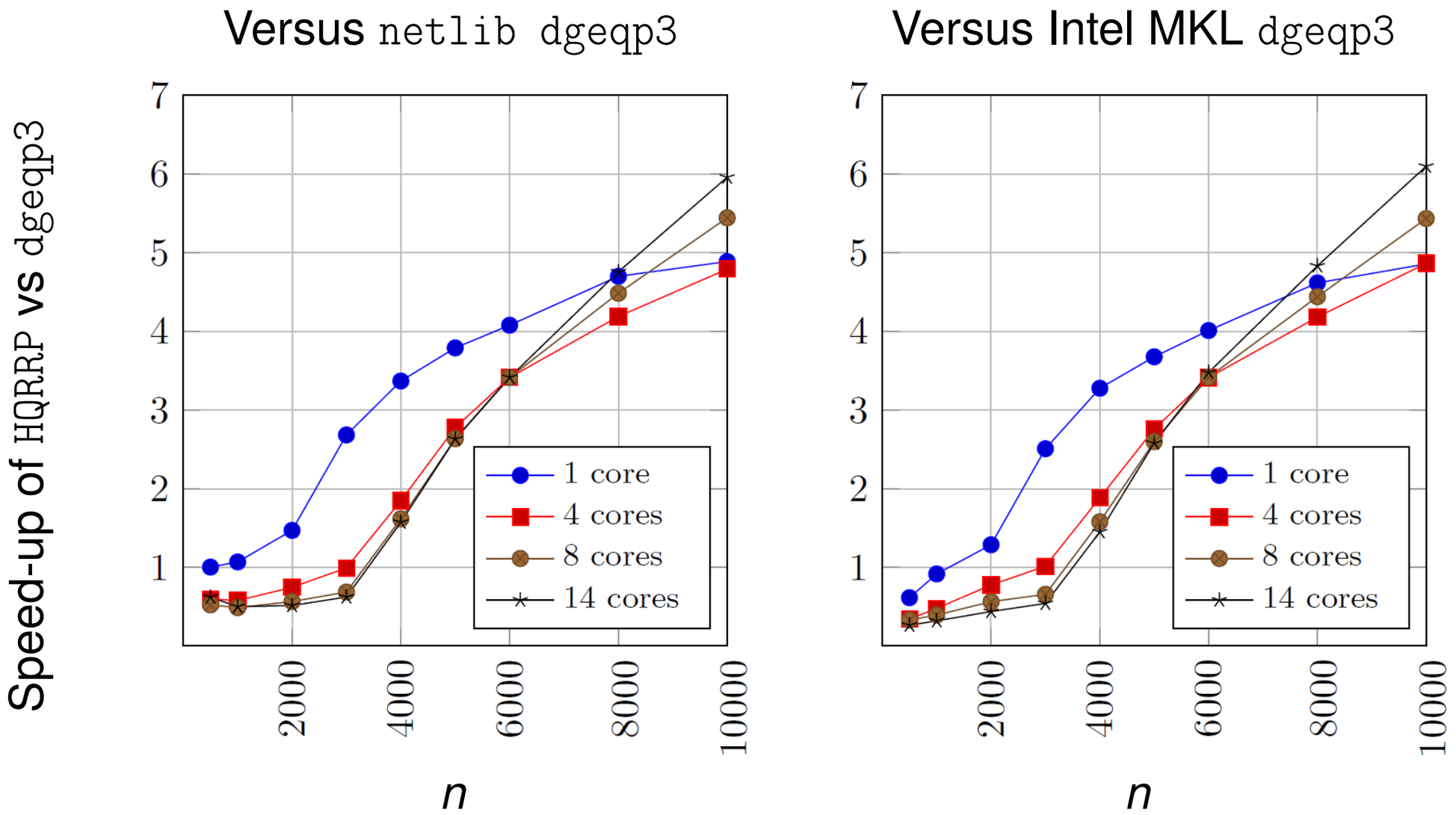
Then we automatically (and almost magically) get an ID of **A**:

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

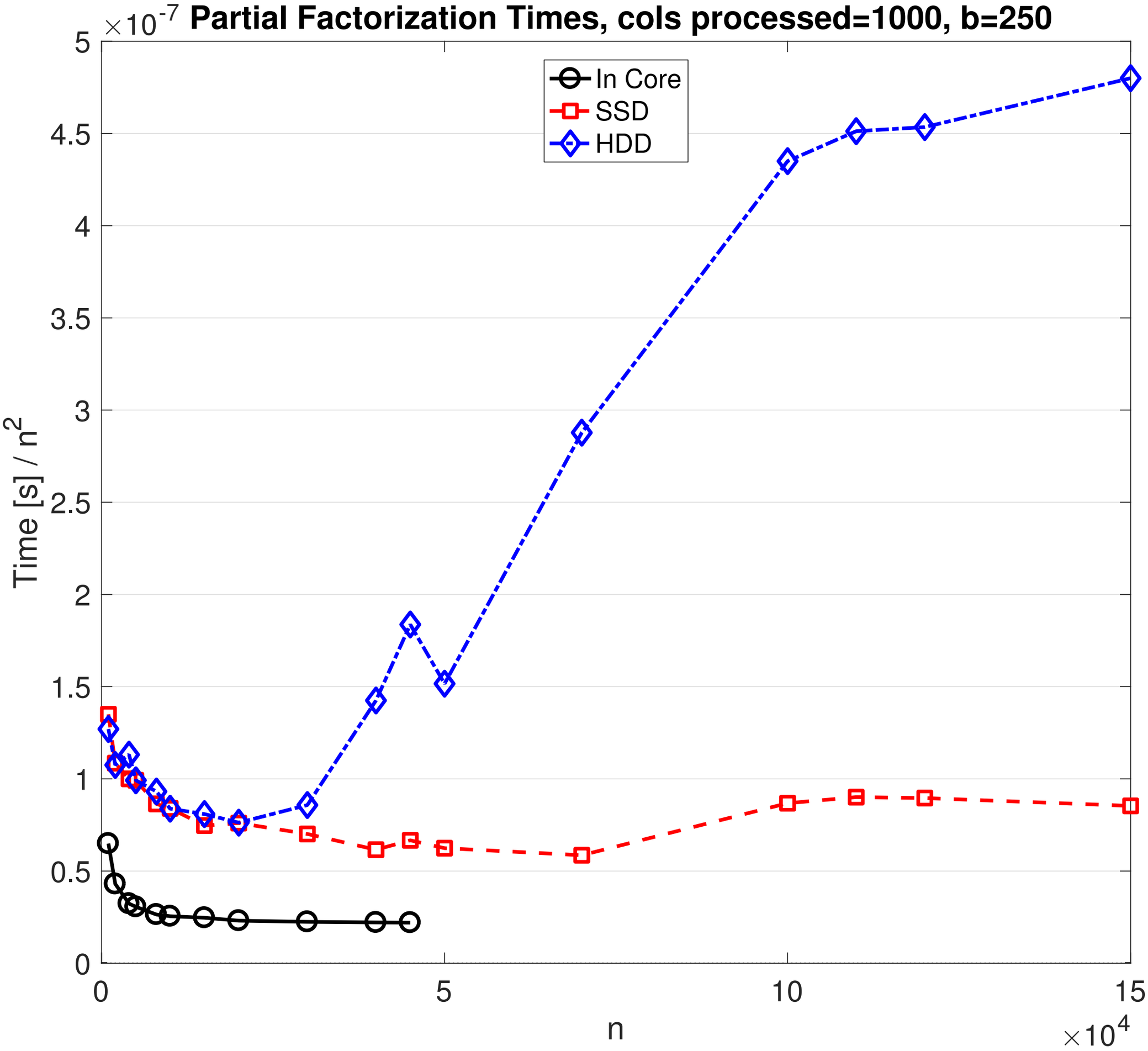
Can be used to compute a CUR decomposition as well.

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)

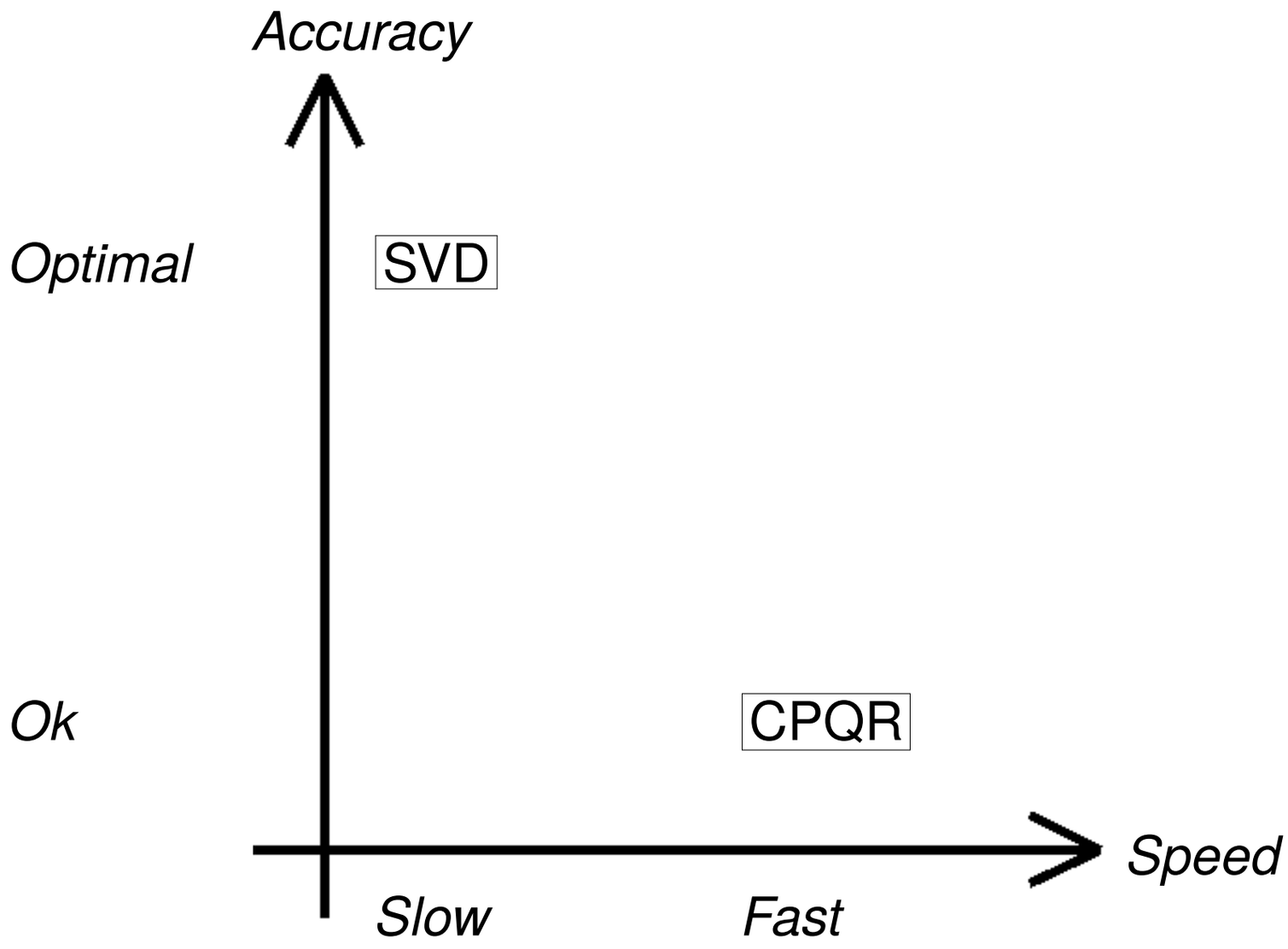


Speedup attained by our randomized algorithm `HQRRP` *for computing a full column pivoted QR factorization of an* $n \times n$ *matrix. The speed-up is measured versus LAPACK's faster routine* `dgeqp3` *as implemented in Netlib (left) and Intel's MKL (right). Our implementation was done in C, and was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

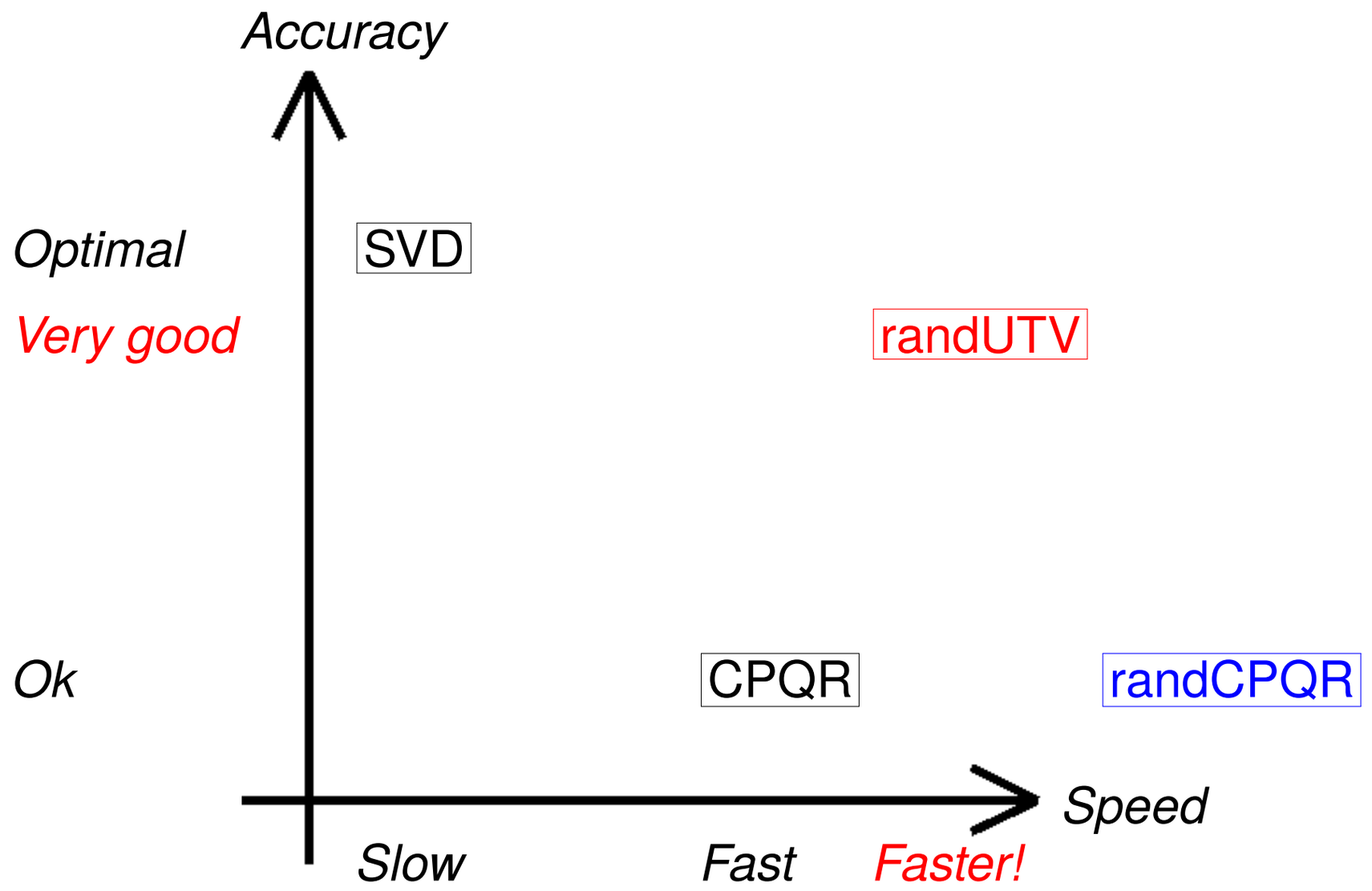
Randomized Column Pivoted QR (randCPQR)

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)

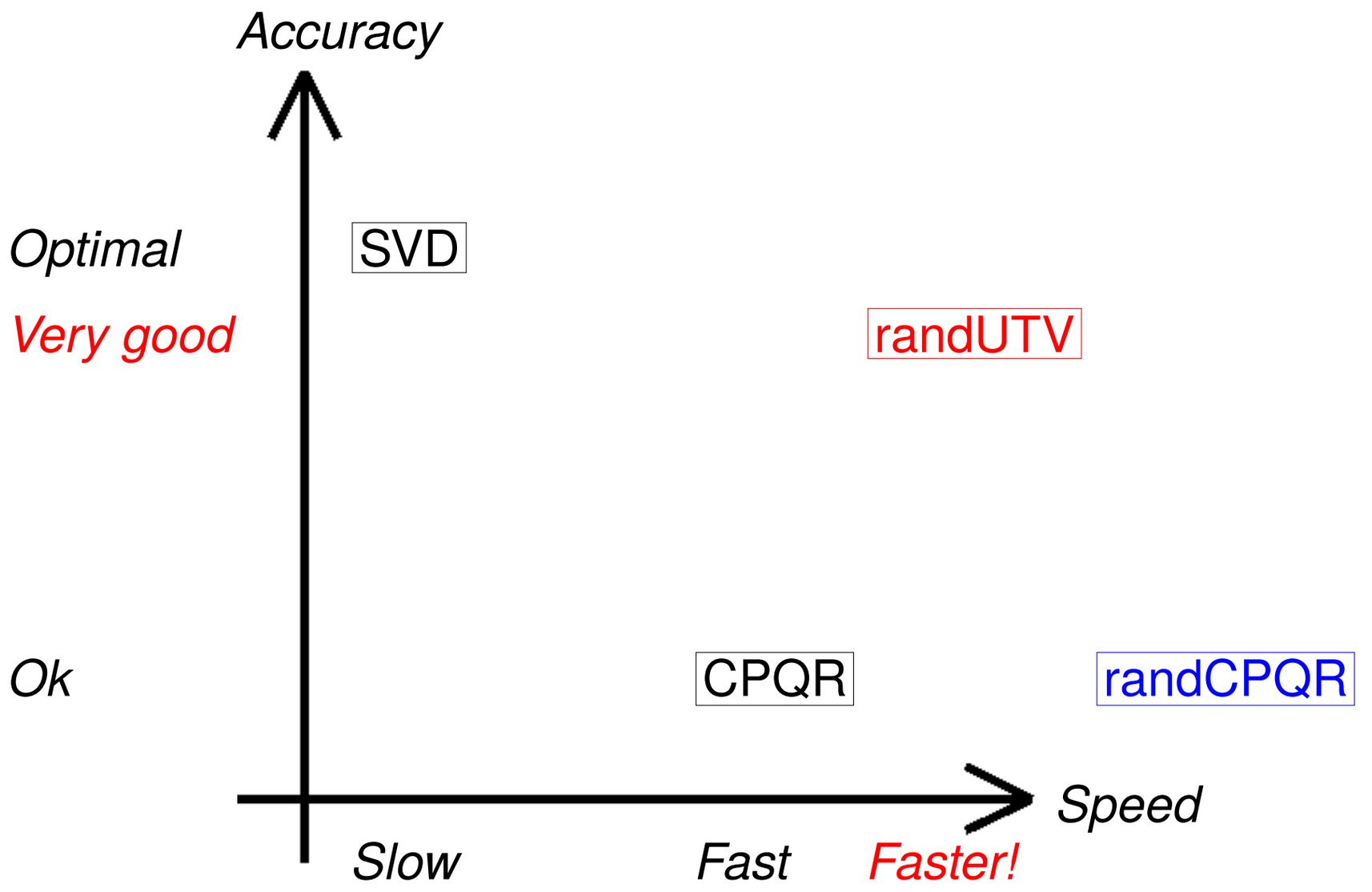
For the task of computing low-rank approximations to matrices, the classical choice is between SVD and column pivoted QR (CPQR). SVD is slow, and CPQR is inaccurate:



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)

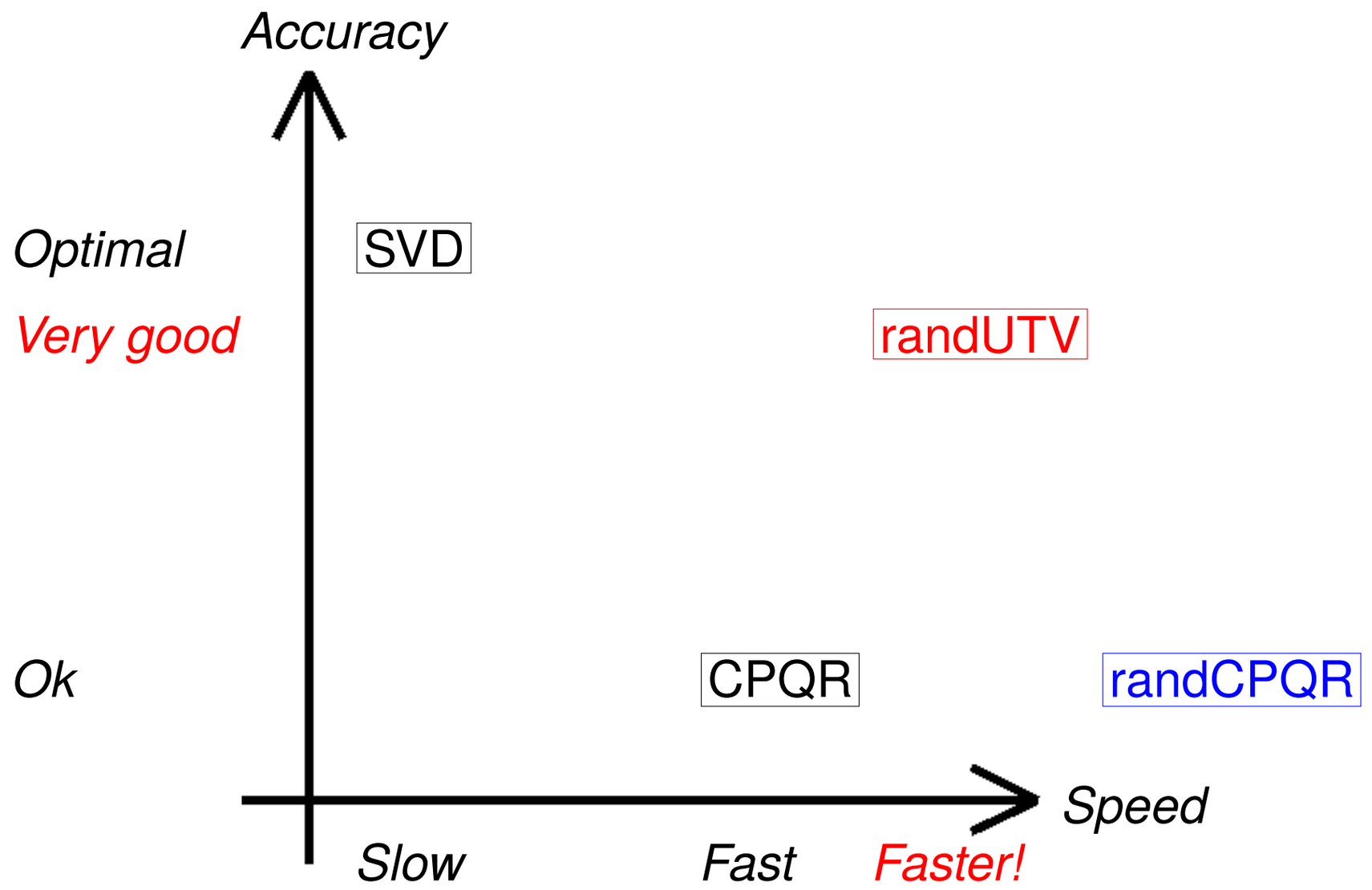
For the task of computing low-rank approximations to matrices, the classical choice is between SVD and column pivoted QR (CPQR). SVD is slow, and CPQR is inaccurate:



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.

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, 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`