Randomised algorithms for solving systems of linear equations

Per-Gunnar Martinsson

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

University of Texas at Austin

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

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

Research support by:

Scope: Let $\mathbf{A}$ be a given $m \times n$ matrix (real or complex), and let $\mathbf{b}$ be a given vector. The talk is about techniques for solving

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

Environments considered: (Time permitting ...)

- $\mathbf{A}$ is square, nonsingular, and stored in RAM.
- $\mathbf{A}$ is square, nonsingular, and stored out of core. (Very large matrix.)
- $\mathbf{A}$ is rectangular, with $m \gg n$. (Very over determined system.)
- $\mathbf{A}$ is a graph Laplacian matrix. (Very large, and sparse).
- $\mathbf{A}$ is an $n \times n$ "kernel matrix", in the sense that given some set of points $\{\boldsymbol{x}_i\}_{i=1}^{n} \subset \mathbb{R}^d$, the ij entry of the matrix can be written as $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some kernel function k.
  - ⋆ Scientific computing: High accuracy required.
  - ⋆ Data analysis: d is large ($d = 4$, or 10, or 100, or 1 000, ...).

Techniques: The recurring theme is *randomisation*.

- Randomized sampling. Typically used to build preconditioners.
- Randomized embeddings. Reduce effective dimension of intermediate steps.

Prelude: We introduce the ideas around *randomized embeddings* by reviewing randomized techniques for low rank approximation. (Recap of Tuesday talk.)

Randomised SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$ of approximate rank k, compute a factorisation

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

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

Randomised SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$ of approximate rank k, compute a factorisation

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

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

The only error we seek to control is

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

We do not aspire to approximate small singular values, to get high *relative* errors, etc.

Randomised SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$ of approximate rank k, compute a factorisation

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

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

Stage A: Build an approximate basis for the range of A: (Using randomisation!)

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

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

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

Stage B: Restrict A to the computed subspace and perform an exact factorisation:

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

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

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

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

Randomised SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$ of approximate rank k, compute a factorisation

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

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

Stage A: Build an approximate basis for the range of A: (Using randomisation!)

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

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

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

Stage B: Restrict A to the computed subspace and perform an exact factorisation:

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

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

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

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

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

Randomised SVD:

Objective: Given an $m \times n$ matrix $\mathbf{A}$ of approximate rank k, compute a factorisation

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

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

Stage A: Build an approximate basis for the range of A: (Using randomisation!)

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

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

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

Stage B: Restrict A to the computed subspace and perform an exact factorisation:

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

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

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

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

Randomised SVD:

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

Output: Rank-$(k + p)$ factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$.

(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 $\mathbf{Y} = \mathbf{A}\mathbf{R}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Randomised SVD:

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

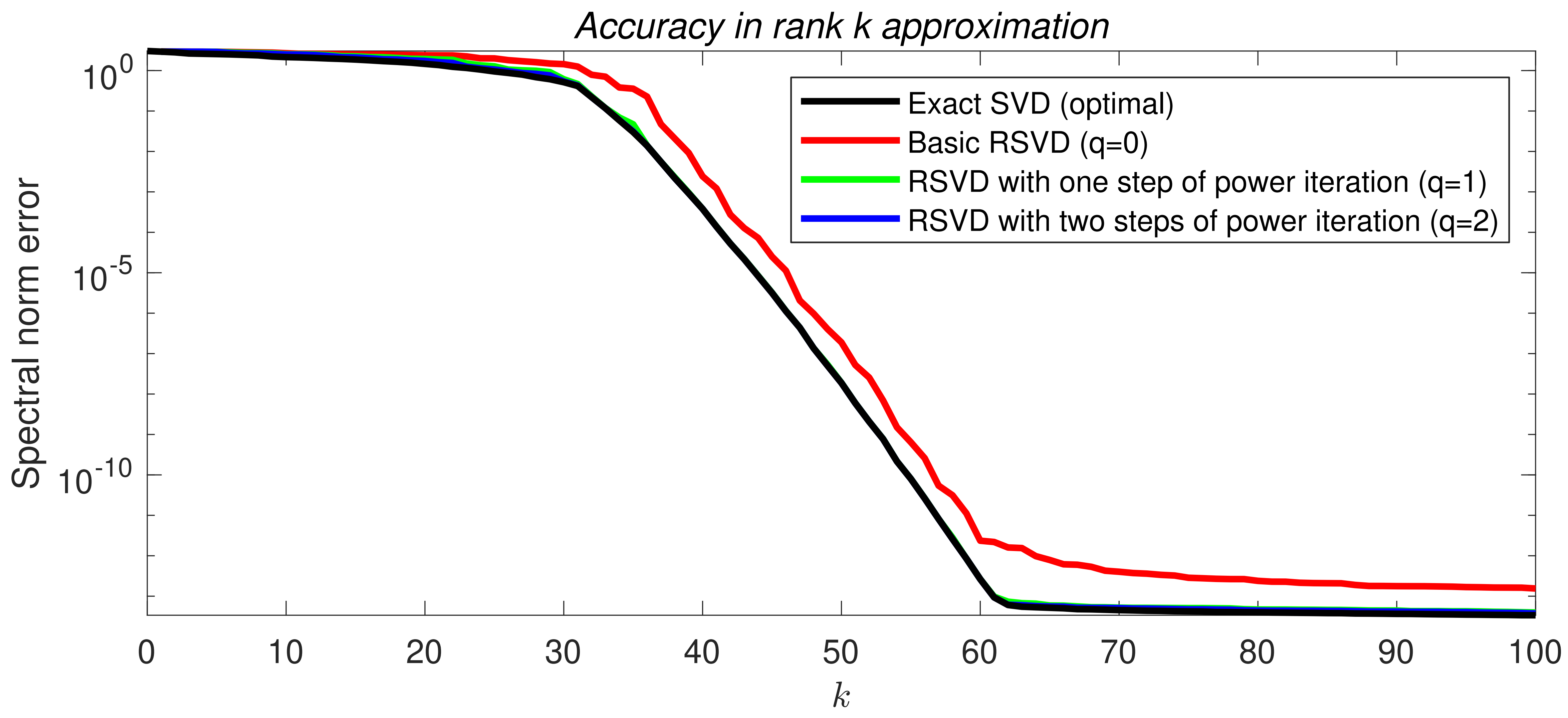
- It is simple to adapt the scheme to the situation where the *tolerance is given,* and the rank has to be determined adaptively.

- The RSVD is in many environments far faster than classical deterministic techniques. The primary reason is that it requires less communication $\rightarrow$ the computational primitive is the *matrix-matrix multiplication*. The method is particularly effective for GPUs, data stored out-of-core, distributed computing, etc.

- A unique advantage of the RSVD is that it can be modified to operate on *streaming data* that cannot be stored at all. “Single-view”.

- Accuracy of the basic scheme is good when the singular values decay reasonably fast. When they do not, the scheme can be combined with Krylov-type ideas: *Taking one or two steps of subspace iteration vastly improves the accuracy.* For instance, use the sampling matrix $\mathbf{Y} = \mathbf{AA}^*\mathbf{AG}$ instead of $\mathbf{Y} = \mathbf{AG}$.

Randomised SVD:



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

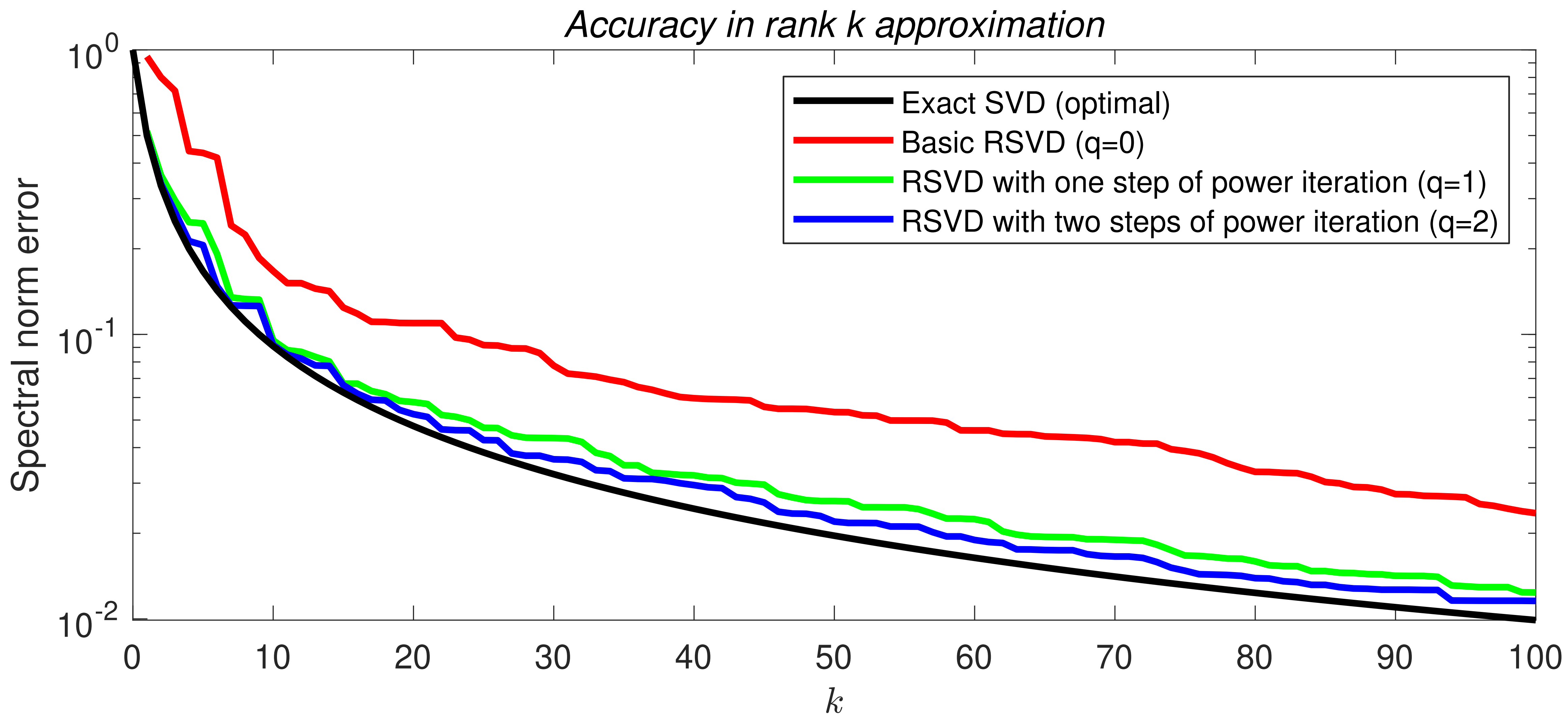
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

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

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

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

Randomised SVD:



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

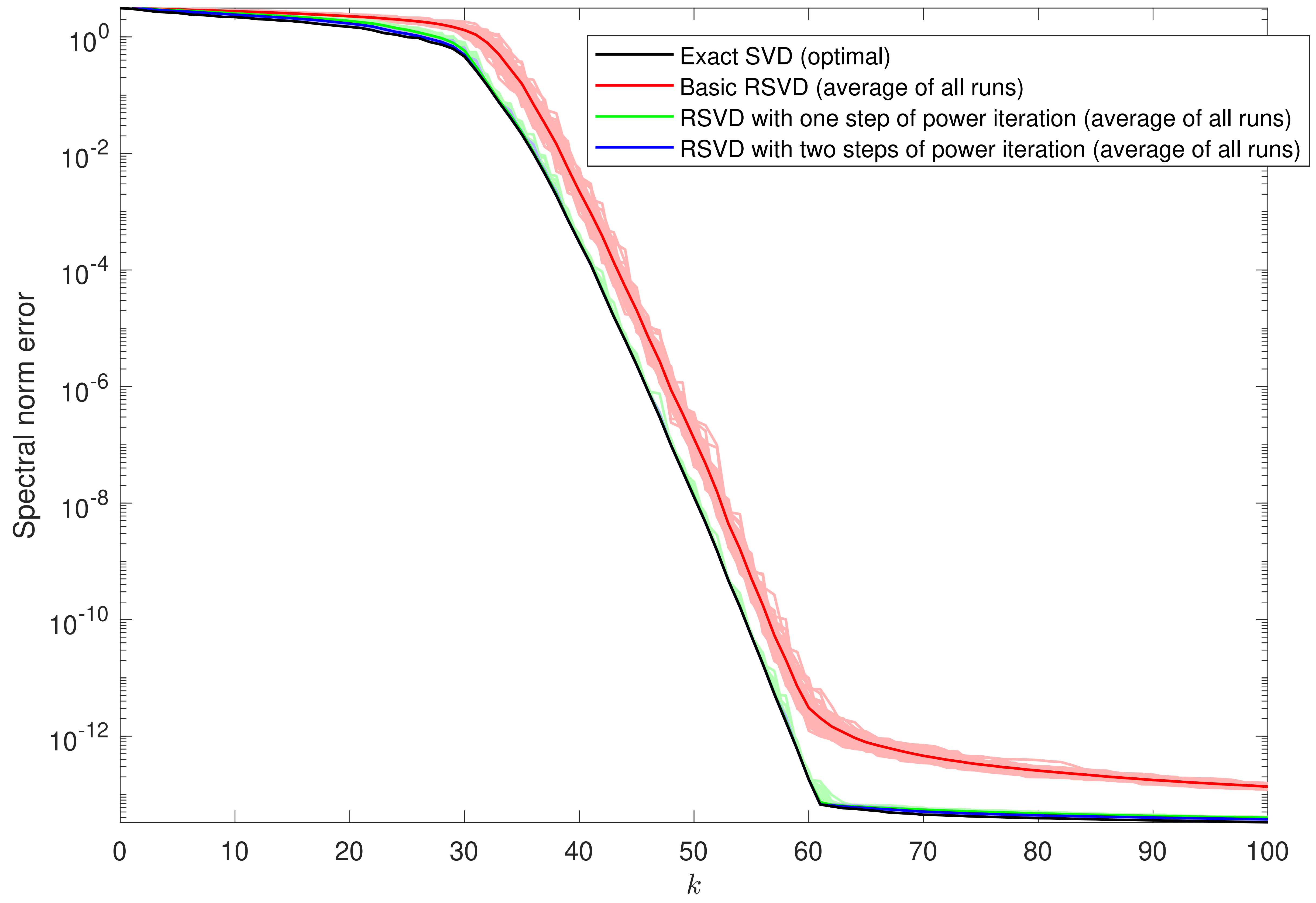
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

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

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

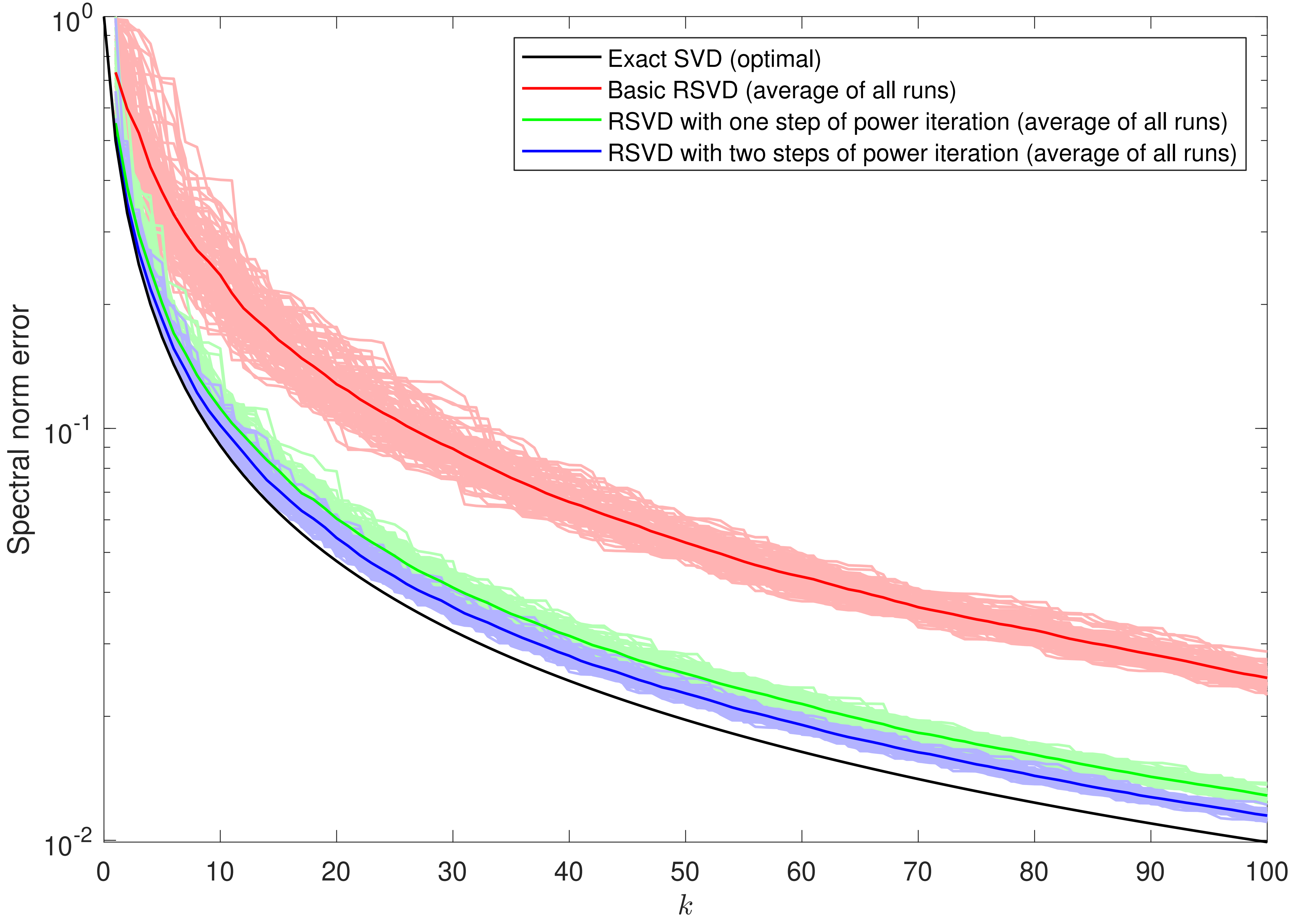
and where $\mathbf{G}$ is a Gaussian random matrix. (Recall that $\mathbf{P}_k\mathbf{A} = \mathbf{U}_k\mathbf{D}_k\mathbf{V}_k^*$.)
The matrix $\mathbf{A}$ now has singular values that decay slowly.

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



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

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



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

Randomised SVD:

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

Output: Rank-$(k + p)$ factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$.

(1) Draw an $n \times (k + p)$ random matrix $\mathbf{R}$.

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

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

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

We can reduce the flop count from $O(mnk)$ to $O(mn\log k)$ by using a so called "fast Johnson-Lindenstrauss" transform. A popular choice is the *subsampled random Fourier Transform (SRFT)* which can be applied using a variation of the FFT. Many other options: sub-sampled Hadamard transform, chains of Givens rotations, sparse projections, …

Randomised SVD:

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

Output: Rank-$(k+p)$ factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$.

(1) Draw an $n \times (k+p)$ random matrix $\mathbf{R}$.

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

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

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

We can reduce the flop count from $O(mnk)$ to $O(mn\log k)$ by using a so called "fast Johnson-Lindenstrauss" transform. A popular choice is the *subsampled random Fourier Transform (SRFT)* which can be applied using a variation of the FFT. Many other options: sub-sampled Hadamard transform, chains of Givens rotations, sparse projections, …

Example: The SRFT takes the form

$$\begin{array}{ccccc} \mathbf{R} & = & \mathbf{D} & \mathbf{F} & \mathbf{S}. \\ n \times k & & n \times n & n \times n & n \times k \end{array}$$

- $\mathbf{D}$ is a diagonal matrix whose entries are i.i.d. random variables drawn from a uniform distribution on the unit circle in $\mathbb{C}$.
- $\mathbf{F}$ is the discrete Fourier transform, $\mathbf{F}_{pq} = \dfrac{1}{\sqrt{n}} e^{-2\pi i (p-1)(q-1)/n}$.
- $\mathbf{S}$ is a matrix whose entries are all zeros except for a single, randomly placed 1 in each column. (So the action of $\mathbf{S}$ is to draw k columns at random from $\mathbf{D}\,\mathbf{F}$.)

Randomised SVD:

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

(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 $\mathbf{Y} = \mathbf{A}\mathbf{R}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{Q}\mathbf{Q}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

We can reduce the flop count from $O(mnk)$ to $O(mn\log k)$ by using a so called "fast Johnson-Lindenstrauss" transform. A popular choice is the *subsampled random Fourier Transform (SRFT)* which can be applied using a variation of the FFT. Many other options: sub-sampled Hadamard transform, chains of Givens rotations, sparse projections, . . .

- The algorithm must be modified a bit beside replacing the random matrix.
- The SRFT leads to large speed-ups *for moderate matrix sizes.*
  For instance, for $m = n = 4000$, and $k \sim 10^2$, we observe about $\times 5$ speedup.
- In practice, accuracy is similar to what you get from Gaussian random matrices.
- Theory is still quite weak.

References: Ailon and Chazelle (2006); Liberty, Rokhlin, Tygert, and Woolfe (2006). Halko, Martinsson, Tropp (2011). Much subsequent work . . .

Linear solvers

Given an $m \times n$ matrix $\mathbf{A}$ (real or complex), we consider the task of solving

$$\mathbf{A}\mathbf{x} = \mathbf{b}.$$

Focus is on the case where $\mathbf{A}$ is of size $n \times n$ and non-singular.

The techniques we describe can be organized as follows:

- $O(n^3)$ methods for general coefficient matrices.
- Faster than $O(n^3)$ methods for general coefficient matrices?
- Linear complexity methods for “special” coefficient matrices.

Linear solvers

Given an $m \times n$ matrix $\mathbf{A}$ (real or complex), we consider the task of solving

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

Focus is on the case where $\mathbf{A}$ is of size $n \times n$ and non-singular.

The techniques we describe can be organized as follows:

- $O(n^3)$ methods for general coefficient matrices.
- Faster than $O(n^3)$ methods for general coefficient matrices?
- Linear complexity methods for "special" coefficient matrices.

Observation: When $\mathbf{A}$ is well-conditioned, iterative methods converge rapidly.
Worst case complexity for solving $\mathbf{Ax} = \mathbf{b}$ to precision ε is then

$$\log\left(\frac{1}{\varepsilon}\right) \times \text{Cost of matrix-vector multiplication.}$$

For a dense matrix, this of course works out to $O(n^2 \log(1/\varepsilon))$.
The challenge concerns matrices that are ill-conditioned.
Or, to be more precise, whose spectra are not clustered.

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

Suppose $\mathbf{A}$ is a dense ill-conditioned matrix of moderate size. In such a case, it is natural to look to $O(n^3)$ methods that compute a full factorisation of the matrix.

Standard options (all with complexity $O(n^3)$) include:

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

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

Suppose **A** is a dense ill-conditioned matrix of moderate size. In such a case, it is natural to look to $O(n^3)$ methods that compute a full factorisation of the matrix.

Standard options (all with complexity $O(n^3)$) include:

• Unpivoted QR (QR) • Partially pivoted LU	• Column pivoted QR (CPQR) • Fully pivoted LU • SVD
Not always stable. Fast.	Always stable. Slow.

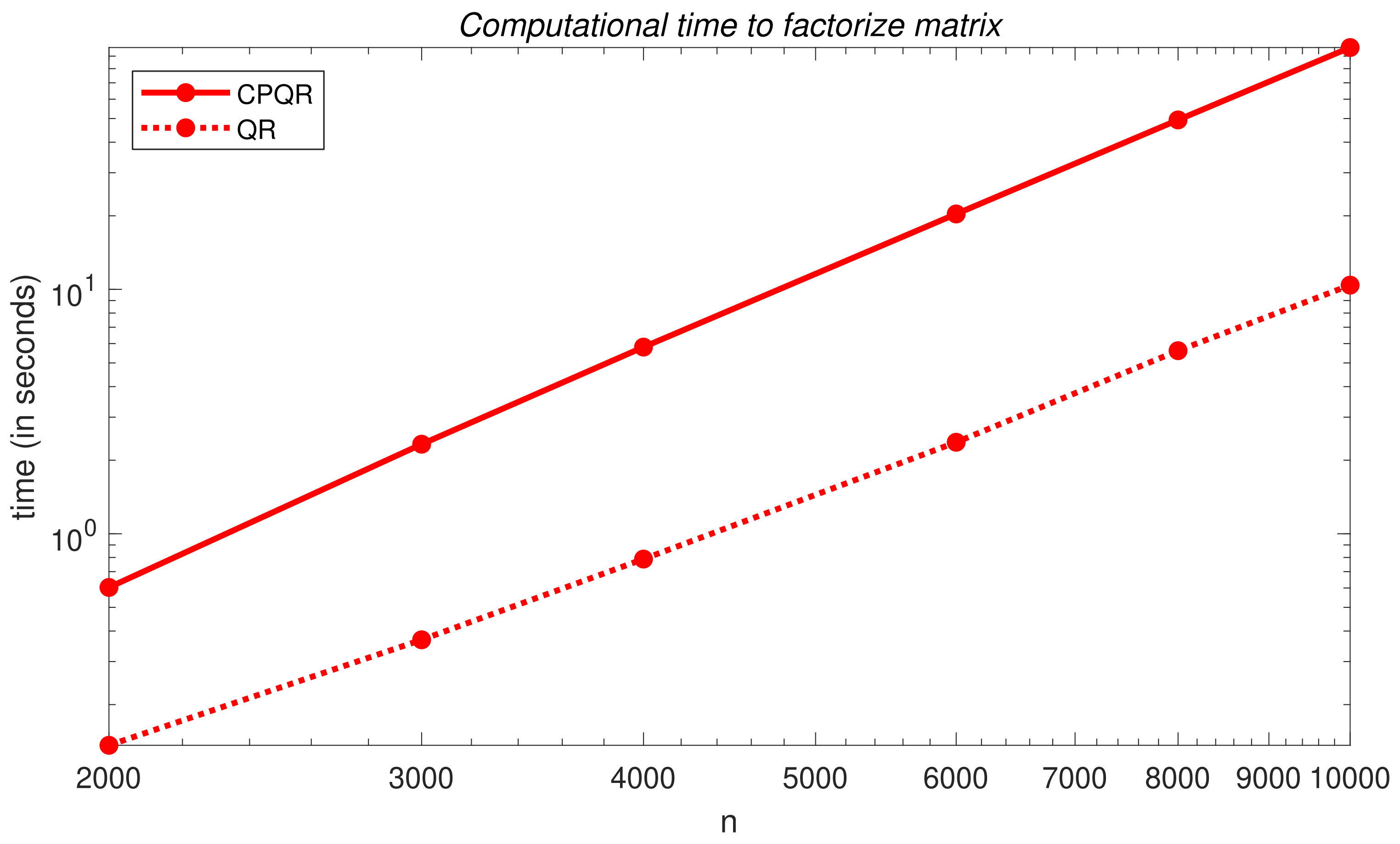
The “robust” factorisations to the right all depend on algorithms that proceed through a sequence of rank-one updates to the matrix. This makes them slow when executed on modern hardware (even on a single core).

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

Suppose **A** is a dense ill-conditioned matrix of moderate size. In such a case, it is natural to look to $O(n^3)$ methods that compute a full factorisation of the matrix.

Standard options (all with complexity $O(n^3)$) include:

• Unpivoted QR (QR) • Partially pivoted LU	• Column pivoted QR (CPQR) • Fully pivoted LU • SVD
Not always stable. Fast.	Always stable. Slow.



Randomised methods for solving **Ax** $=$ **b**: $O(n^3)$ complexity methods

The culprit preventing us from attaining high performance is *pivoting* since it relies on a sequence of rank-one updates.

Randomised methods for solving **Ax** = **b**: $O(n^3)$ complexity methods

The culprit preventing us from attaining high performance is *pivoting* since it relies on a sequence of rank-one updates.

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

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

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

(2) Perform unpivoted QR on the new matrix

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

The resulting factorisation

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

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.

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

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

1. Draw an $n \times n$ Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = \left(\mathbf{AA}^*\right)^q\mathbf{G}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{VR}_{\mathrm{trash}}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorisation

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

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

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

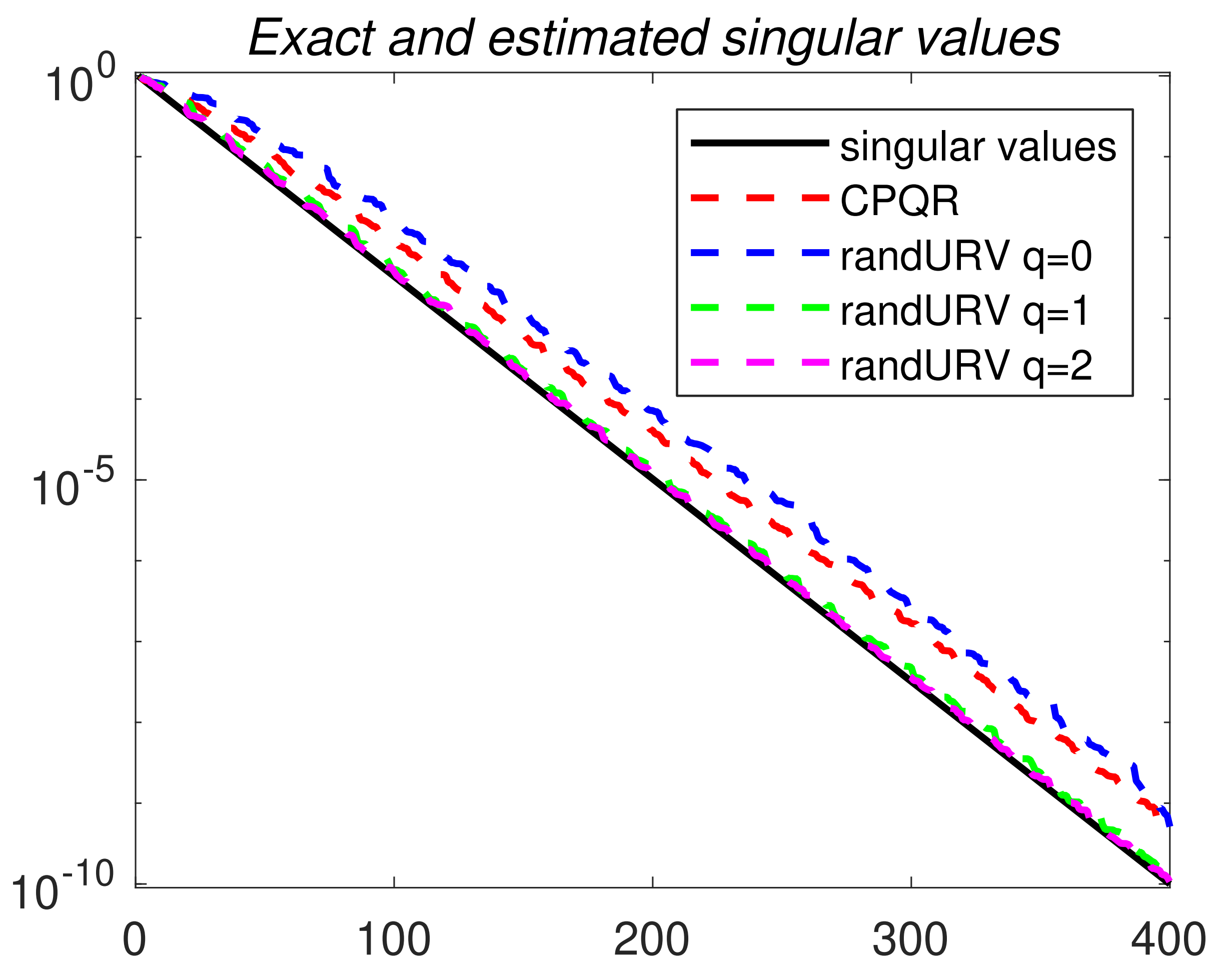
Improved URV factorisation: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

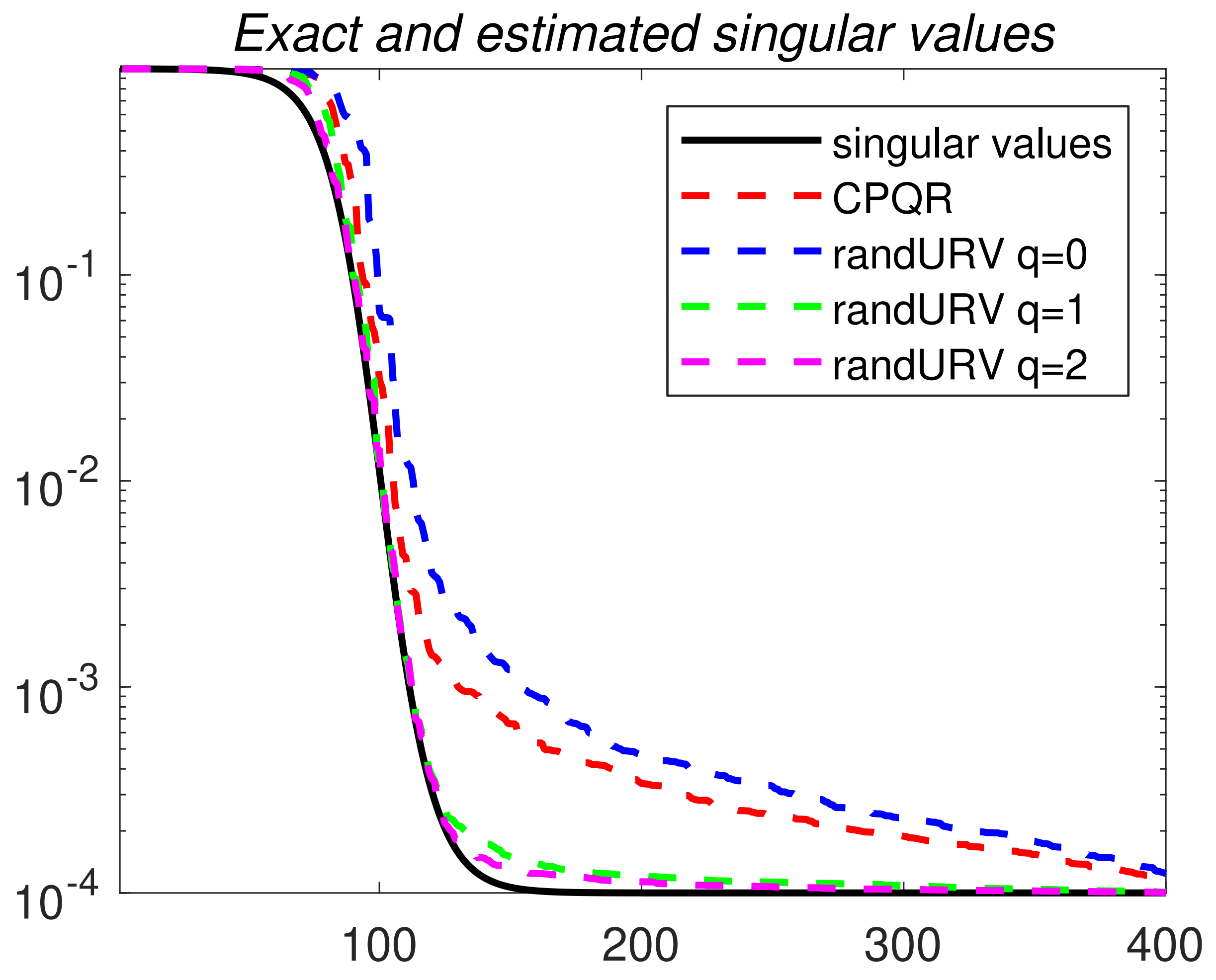
1. Draw an $n \times n$ Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = (\mathbf{AA}^*)^q\mathbf{G}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{VR}_{\rm trash}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorisation

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

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





Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

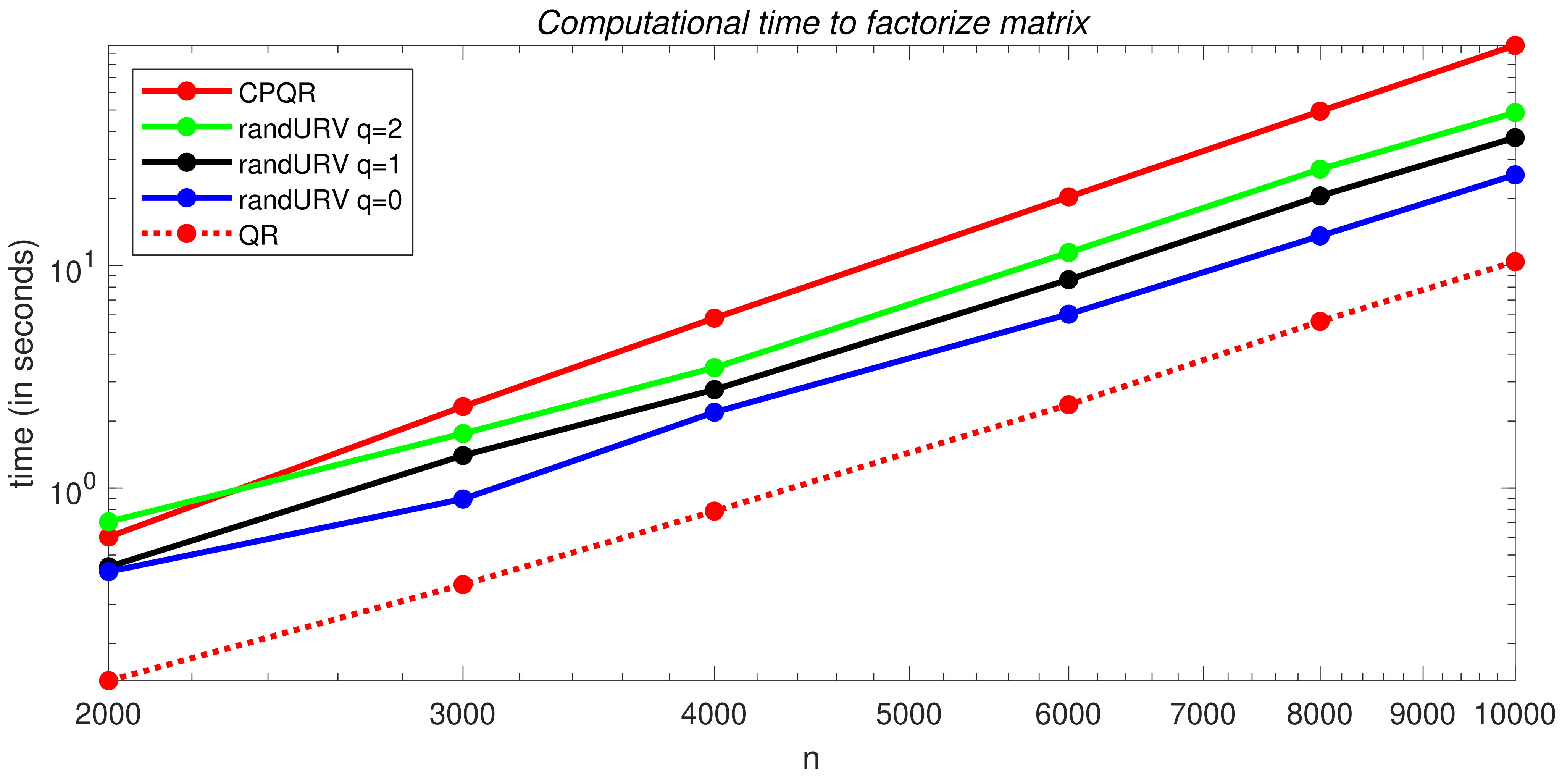
Improved URV factorisation: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw an $n \times n$ Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = \left(\mathbf{AA}^*\right)^q\mathbf{G}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{VR}_{\rm trash}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorisation

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

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



Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

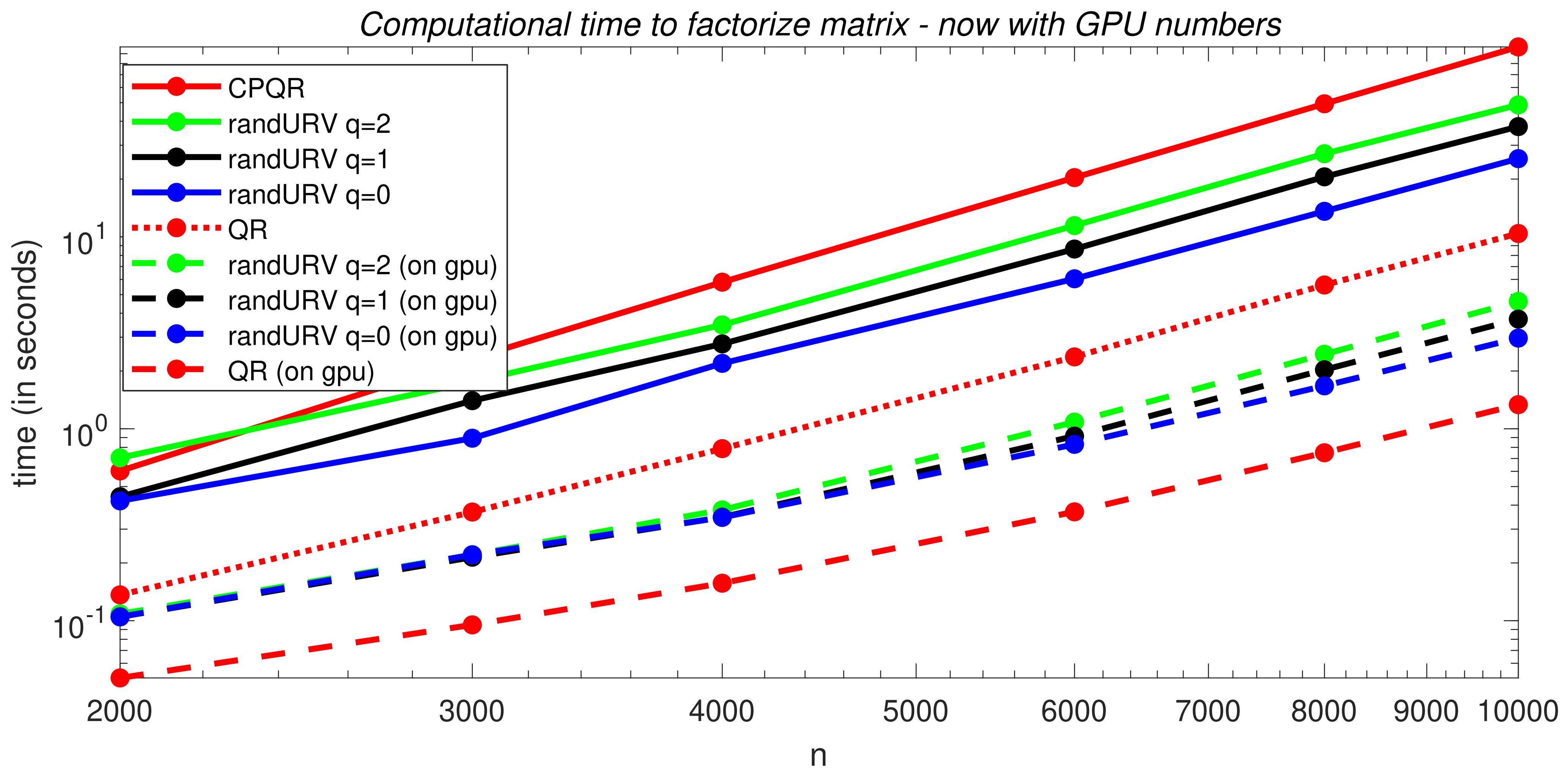
Improved URV factorisation: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

1. Draw an $n \times n$ Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = \left(\mathbf{AA}^*\right)^q \mathbf{G}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{VR}_{\rm trash}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorisation

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

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



Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

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

1. Draw an $n \times n$ Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = \left(\mathbf{AA}^*\right)^q \mathbf{G}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{VR}_{\rm trash}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorisation

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

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);
```

Reference: *The PowerURV algorithm for computing rank-revealing full factorizations*

Abinand Gopal, Per-Gunnar Martinsson, arxiv:1812.06007.

But . . .

But . . . the times for CPQR refer to classical deterministic CPQR.

It turns out that we can greatly accelerate this computation through randomisation.

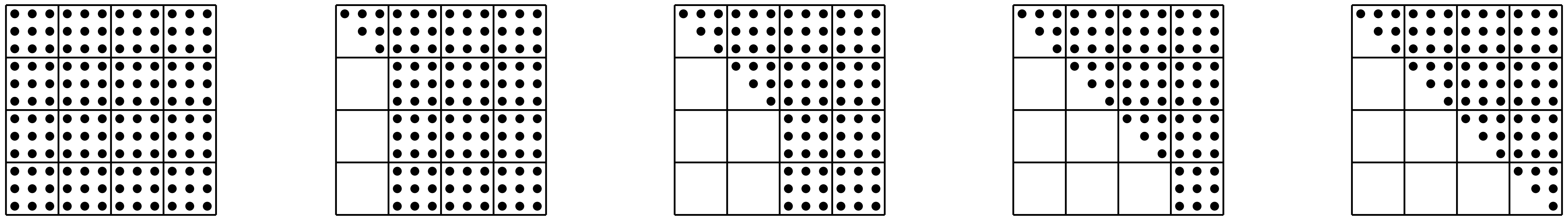
Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

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

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

where, as usual, **Q** should be ON, **P** is a permutation, and **R** is upper triangular.

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



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

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

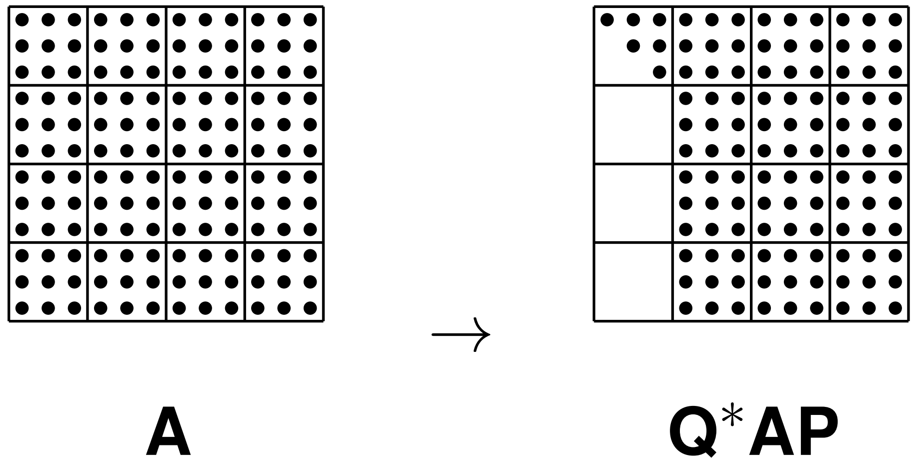
The key challenge has been to find good permutation matrices.
We seek $\mathbf{P}_j$ so that the set of b chosen columns *has maximal spanning volume.*

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

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

How to do block pivoting using randomisation:

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



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

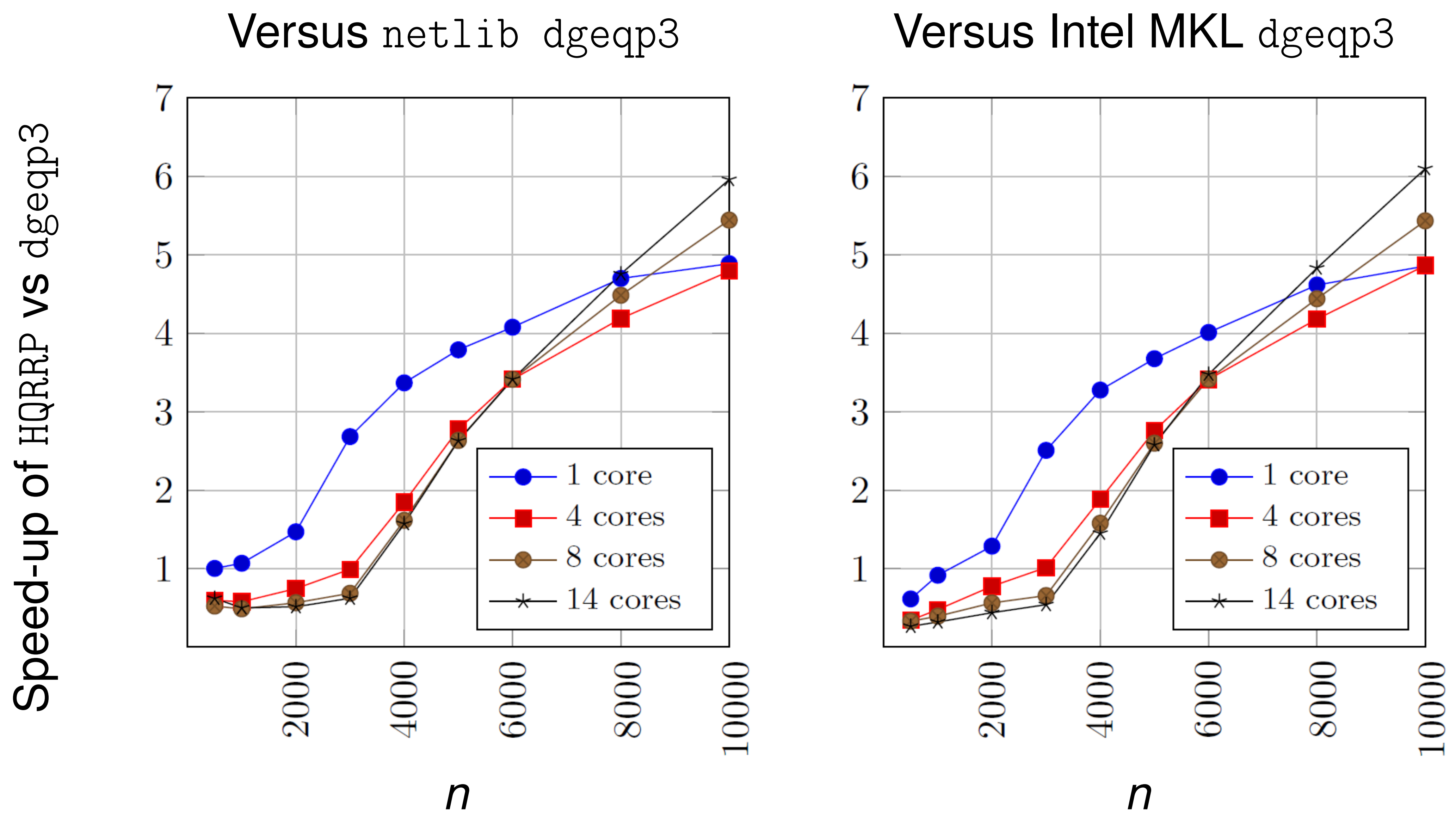
$$\begin{array}{ccccc} \mathbf{Y} & = & \mathbf{G} & & \mathbf{A} \\ b \times n & & b \times m & & m \times n \end{array}$$

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**:

$$\begin{array}{ccccc} \mathbf{Y} & \mathbf{P} & = & \mathbf{Q}_{\rm trash} & \mathbf{R}_{\rm trash} \\ b \times n & n \times n & & b \times b & b \times n \end{array}$$

Randomised methods for solving Ax = b: $O(n^3)$ complexity methods



Speedup attained by our randomised algorithm HQRRP for computing a full column pivoted QR factorisation 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/

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

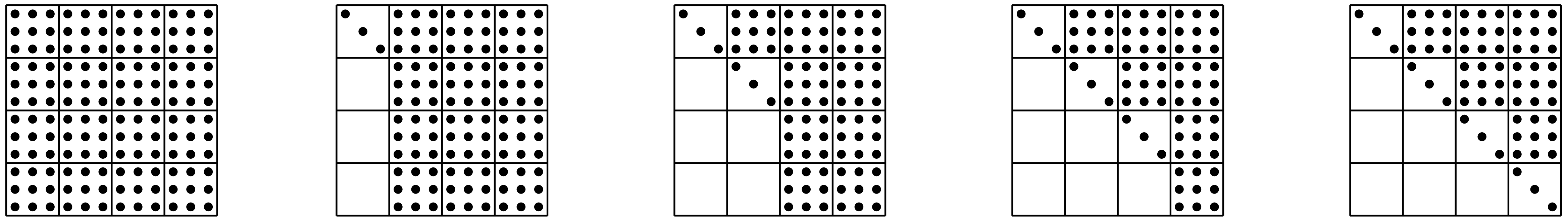
Given a dense $n \times n$ matrix $\mathbf{A}$, compute a factorisation

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

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.

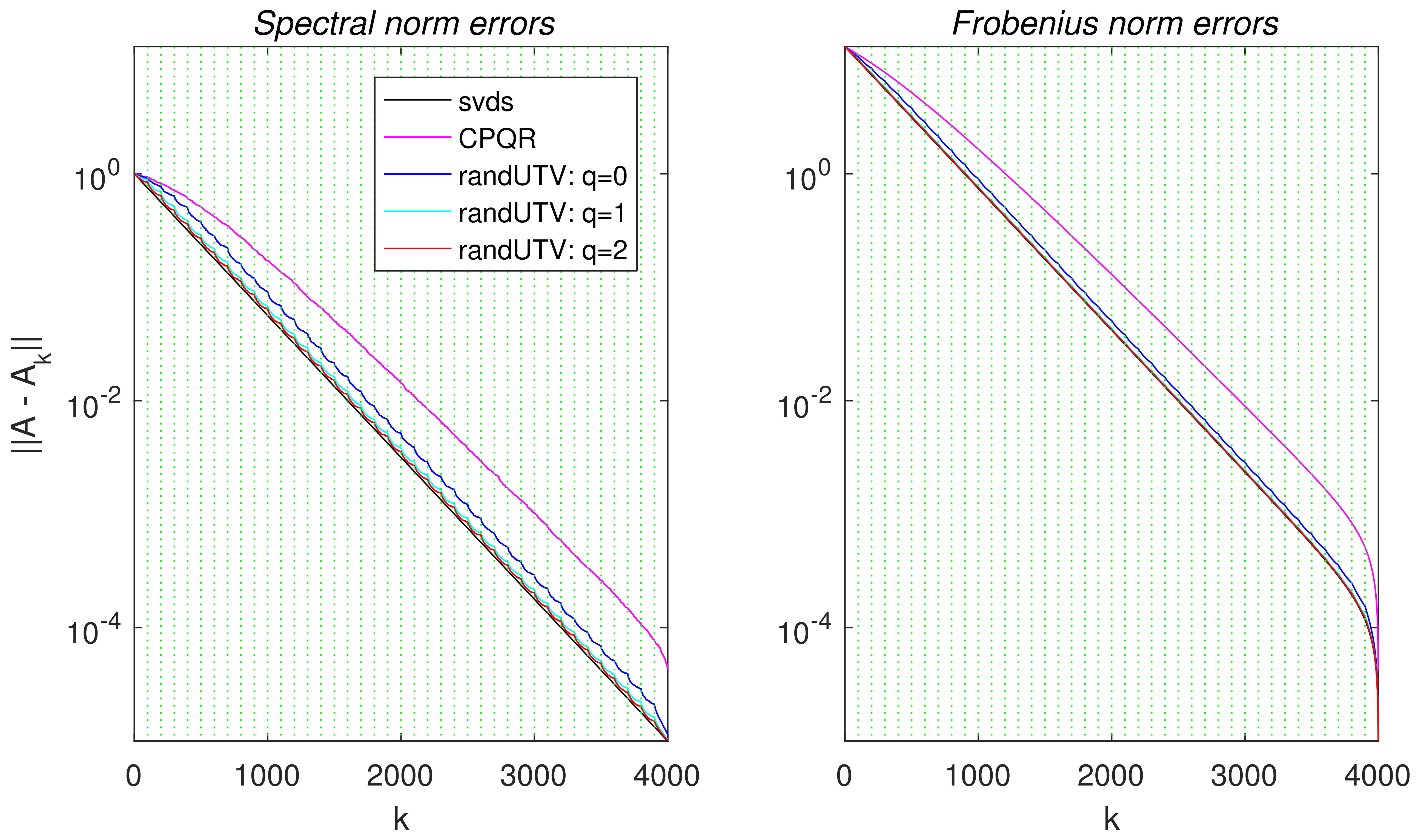
The technique proposed is based on a blocked version of classical Householder QR:



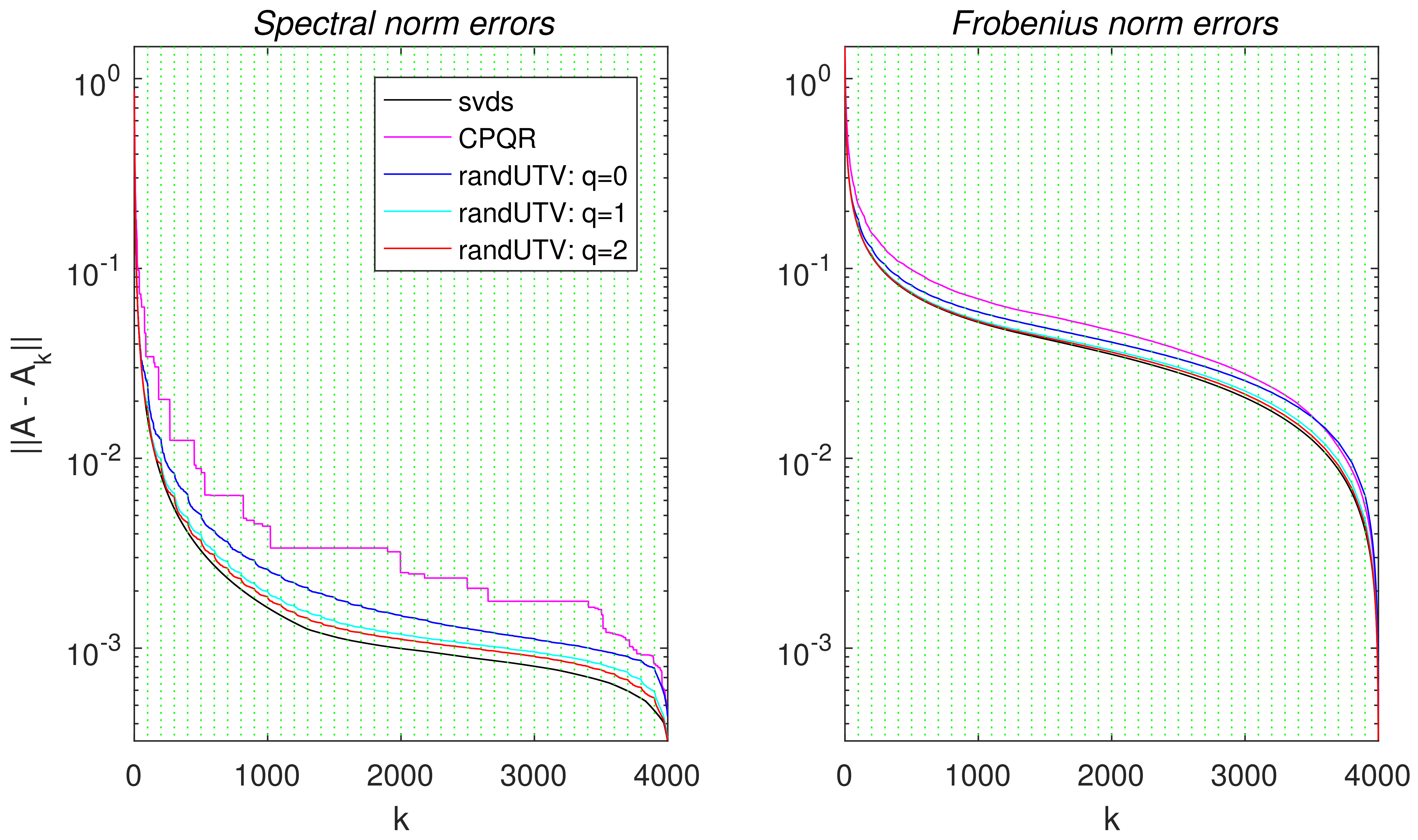
$\mathbf{A}_0 = \mathbf{A}$ $\quad$ $\mathbf{A}_1 = \mathbf{U}_1^*\mathbf{A}_0\mathbf{V}_1$ $\quad$ $\mathbf{A}_2 = \mathbf{U}_2^*\mathbf{A}_1\mathbf{V}_2$ $\quad$ $\mathbf{A}_3 = \mathbf{U}_3^*\mathbf{A}_2\mathbf{V}_3$ $\quad$ $\mathbf{A}_4 = \mathbf{U}_4^*\mathbf{A}_3\mathbf{V}_4$

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

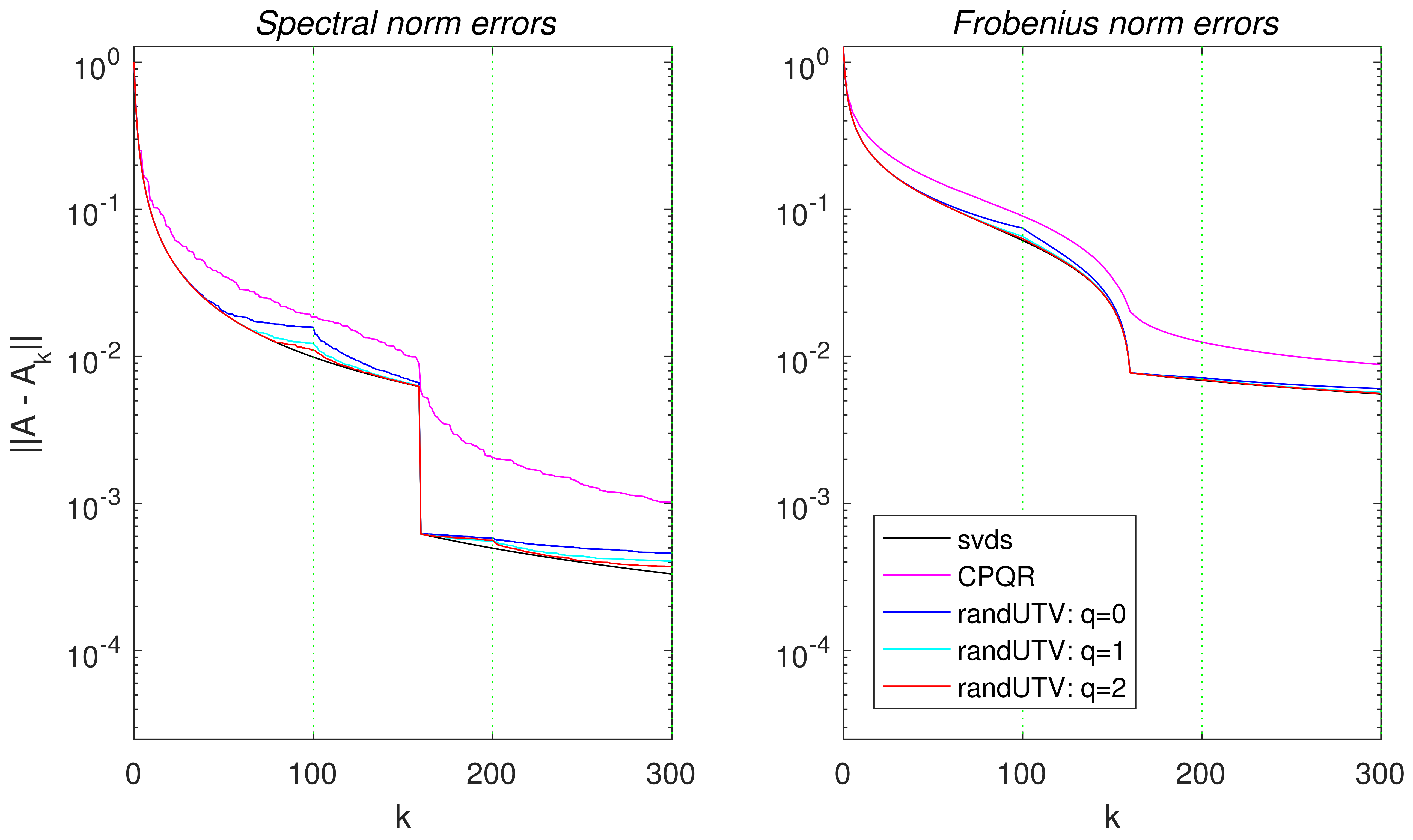
Our objective is in each step to find an approximation *to the linear subspace* spanned by the b dominant singular vectors of a matrix. The randomised range finder is perfect for this, especially when a small number of power iterations are performed. Easier and more natural than choosing pivoting vectors.



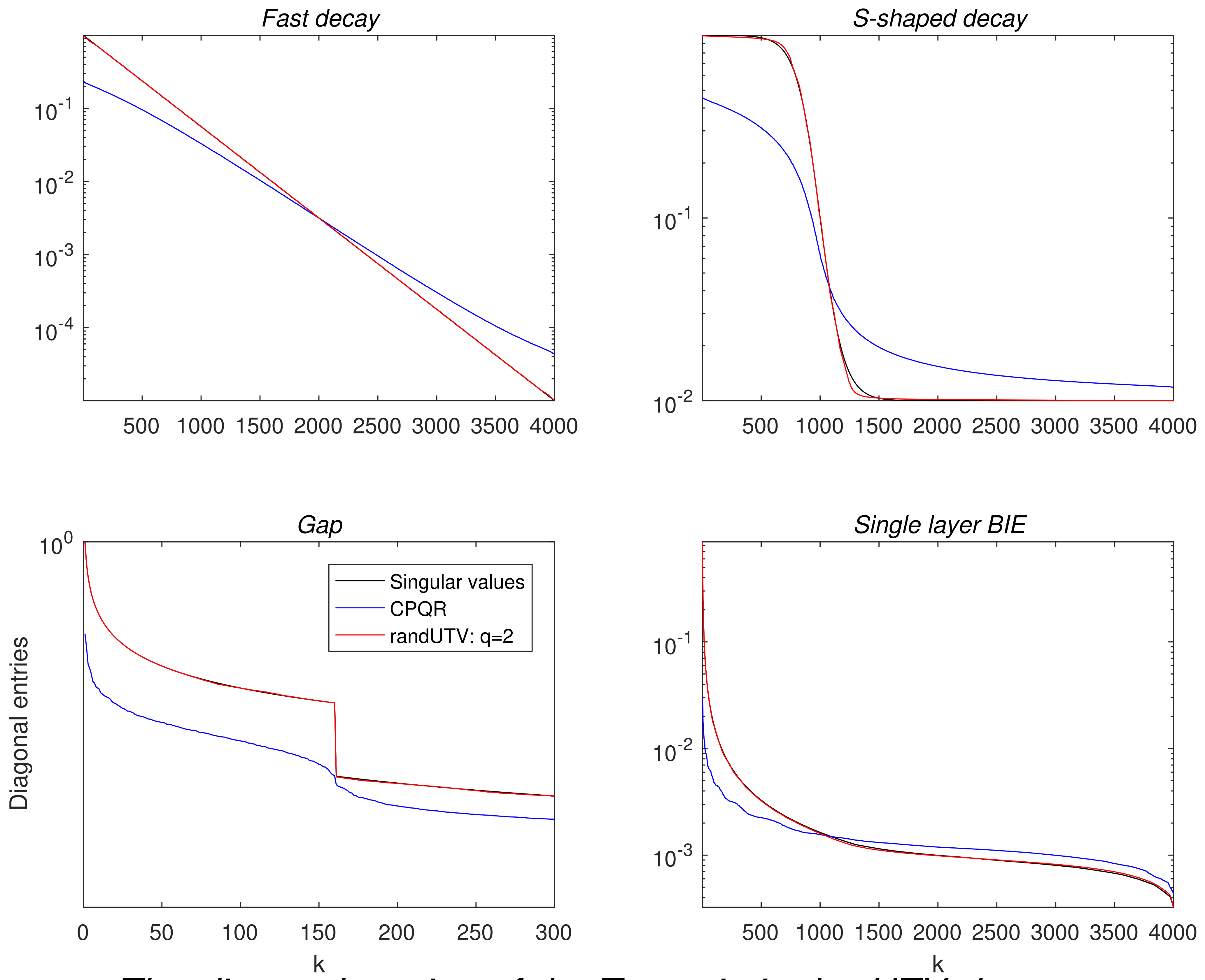
Rank-k approximation errors for the matrix "Fast Decay" of size 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*



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.*



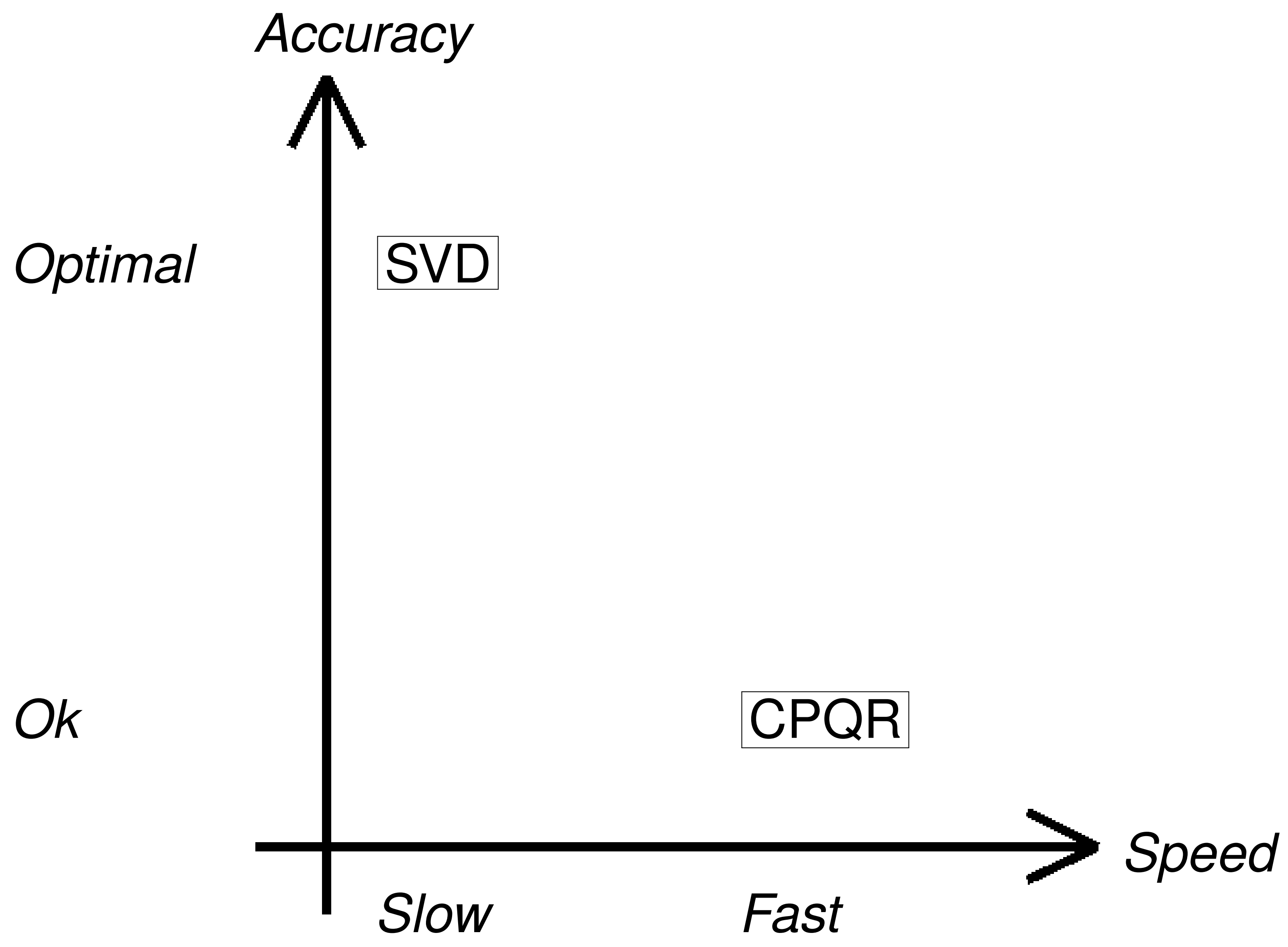
Rank-k approximation errors for $k \leq 300$ for the matrix "Gap" of size 4000×4000. The black lines mark the theoretically minimal errors. The block size was $b = 100$ and the green vertical lines mark block limits.



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

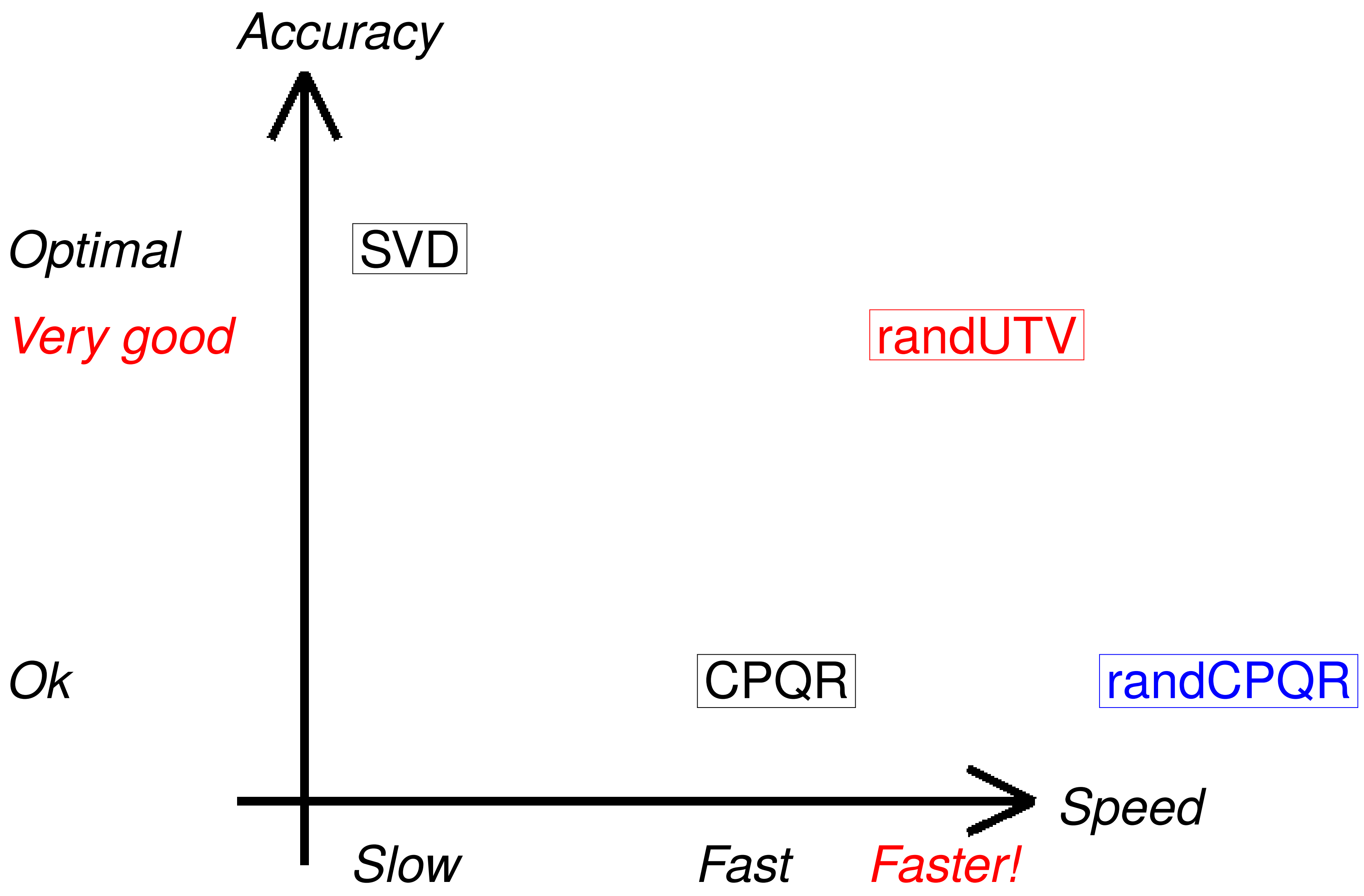
Randomised methods for solving Ax $=$ b: $O(n^3)$ complexity methods

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:



Randomised methods for solving Ax $=$ b: $O(n^3)$ complexity methods

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

Randomised methods for solving $Ax = b$: Strassen-type methods

The essential feature of the randomised 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:

Randomisation allows you to use “fast” matrix-matrix multiplication algorithms to compute rank-revealing factorisations in a numerically stable way. In particular:

$$\text{fast+stable matrix-matrix multiplication} \quad \Rightarrow \quad \text{fast+stable linear system solve}$$

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

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: RSVD as pre-conditioner

Let us consider $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{A} \in \mathbb{R}^{n \times n}$ a symmetric positive definite matrix.

A standard solution technique here is conjugate gradients (CG). The error at step k is known to converge to zero with at least the speed $O(\gamma^k)$ where

$$\gamma = \frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1},$$

and where $\kappa(\mathbf{A})$ is the condition number of $\mathbf{A}$.

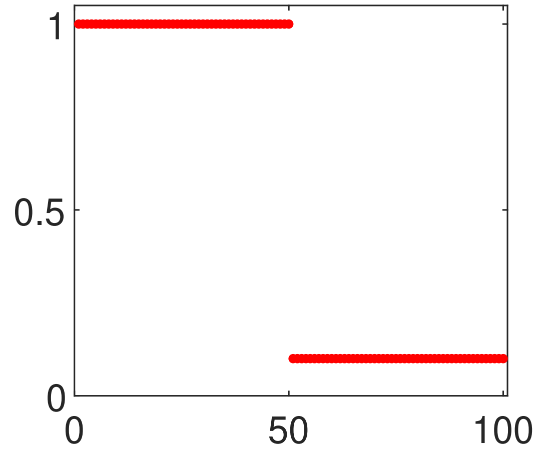
Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: RSVD as pre-conditioner

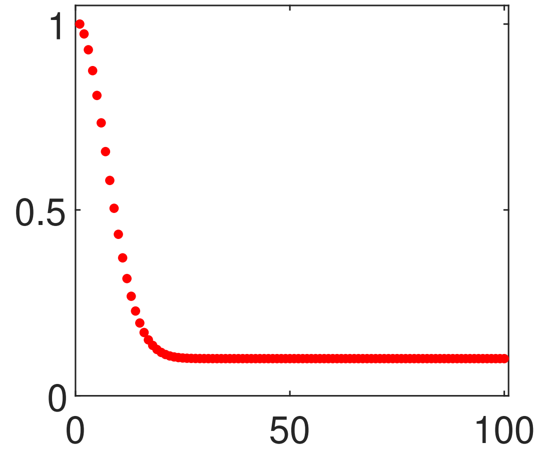
Let us consider $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{A} \in \mathbb{R}^{n\times n}$ a symmetric positive definite matrix.

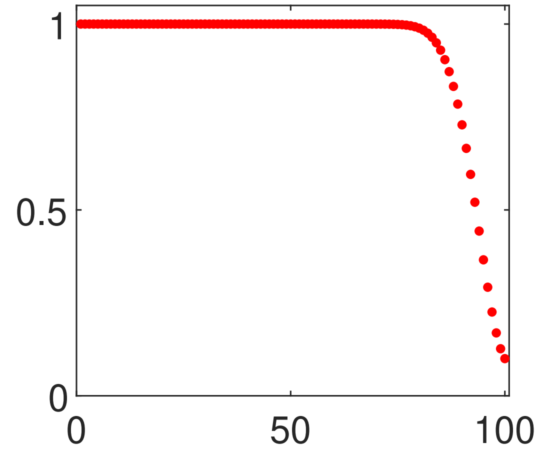
A standard solution technique here is conjugate gradients (CG). The error at step k is known to converge to zero with at least the speed $O(\gamma^k)$ where

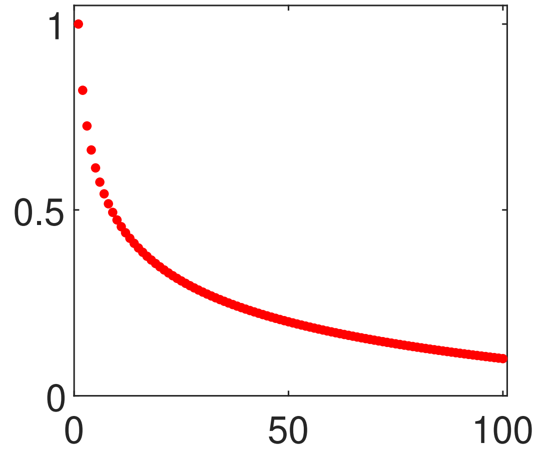
$$\gamma = \frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1},$$

and where $\kappa(\mathbf{A})$ is the condition number of $\mathbf{A}$. But the *clustering* of the spectrum, matters! Consider four spectra with $\lambda_{\max}/\lambda_{\min} = 10$:


(a)


(b)


(c)


(d)

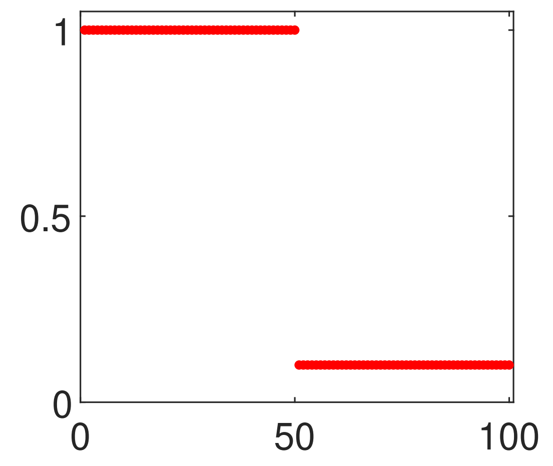
(a) CG converges to the exact answer after 2 iterations.

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: RSVD as pre-conditioner

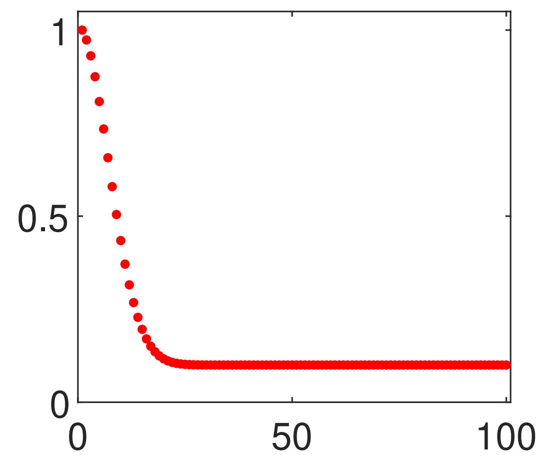
Let us consider $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{A} \in \mathbb{R}^{n\times n}$ a symmetric positive definite matrix.
A standard solution technique here is conjugate gradients (CG). The error at step k is known to converge to zero with at least the speed $O(\gamma^k)$ where

$$\gamma = \frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1},$$

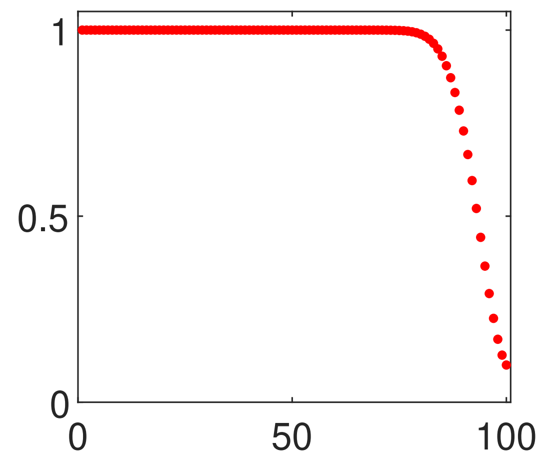
and where $\kappa(\mathbf{A})$ is the condition number of $\mathbf{A}$. But the *clustering* of the spectrum, matters! Consider four spectra with $\lambda_{\max}/\lambda_{\min} = 10$:



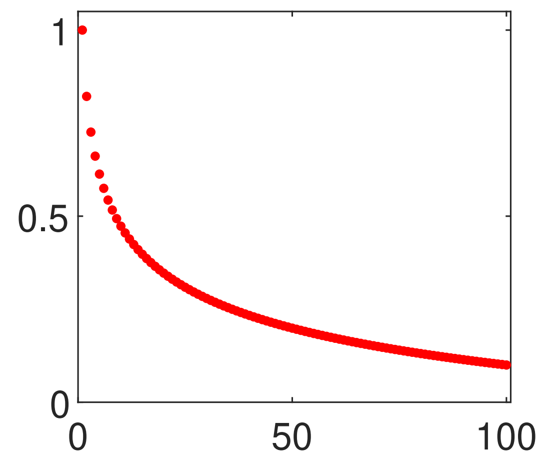
(a)



(b)



(c)



(d)

(b) RSVD provides a preconditioner! Suppose $\mathbf{A} \approx \mathbf{UDU}^*$ captures the k largest eigenmodes. Then use

$$\mathbf{M} = \frac{1}{\lambda_{k+1}^{\text{approx}}}\mathbf{UDU} + \left(\mathbf{I} - \mathbf{UDU}^*\right).$$

as a preconditioner to “attenuate” the outlying large eigenvalues.

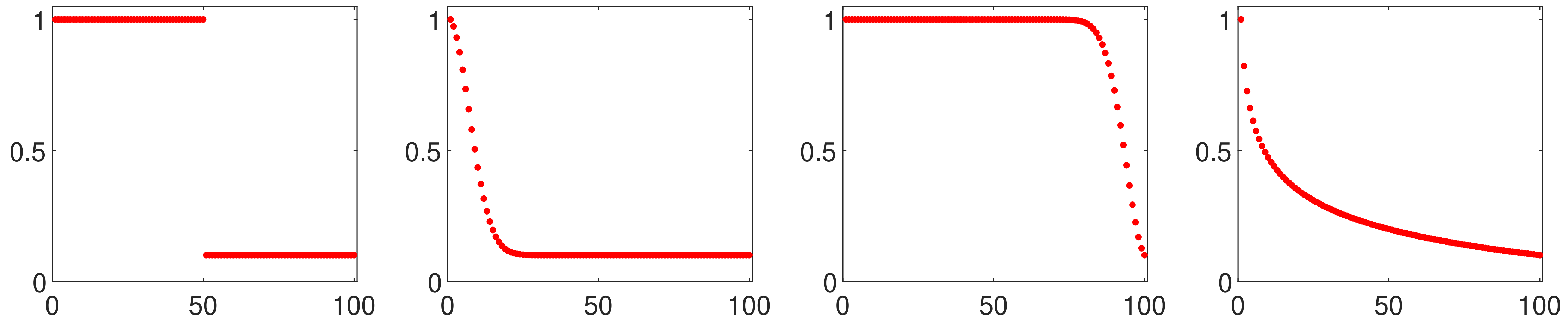
Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: RSVD as pre-conditioner

Let us consider $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{A} \in \mathbb{R}^{n\times n}$ a symmetric positive definite matrix.

A standard solution technique here is conjugate gradients (CG). The error at step k is known to converge to zero with at least the speed $O(\gamma^k)$ where

$$\gamma = \frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1},$$

and where $\kappa(\mathbf{A})$ is the condition number of $\mathbf{A}$. But the *clustering* of the spectrum, matters! Consider four spectra with $\lambda_{\max}/\lambda_{\min} = 10$:



(a) (b) (c) (d)

(c) & (d): Subject of current research . . .

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomised pre-conditioning

Let us consider

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

for $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m \gg n$. Complexity of standard solvers: $O(mn^2)$

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomised pre-conditioning

Let us consider

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

for $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \gg n$. Complexity of standard solvers: $O(mn^2)$

Method proposed by Rokhlin and Tygert (PNAS 2008): Form a “sketched equation”

$$\mathbf{X}^*\mathbf{Ax} = \mathbf{X}^*\mathbf{b}$$

where $\mathbf{X}$ is an $m \times \ell$ SRFT. Compute QR factorisation of the new coefficient matrix

$$\mathbf{X}^*\mathbf{A} = \mathbf{QRP}^*.$$

Form a preconditioner

$$\mathbf{M} = \mathbf{RP}^*.$$

Solve the preconditioned linear system

$$\big(\mathbf{AM}^{-1}\big)\ \underbrace{\big(\mathbf{Mx}\big)}_{=:\mathbf{y}} = \mathbf{b}$$

for the new unknown $\mathbf{y}$. Complexity of randomised solver: $O\big((\log(n) + \log(1/\varepsilon))mn + n^3\big)$.

Later improvements include BLENDENPIK by Avron, Maymounkov, Toledo (2010).

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomised pre-conditioning

Let us consider

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

for $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \gg n$. Complexity of standard solvers: $O(mn^2)$

Method proposed by Rokhlin and Tygert (PNAS 2008): Form a “sketched equation”

$$\mathbf{X}^*\mathbf{Ax} = \mathbf{X}^*\mathbf{b}$$

where $\mathbf{X}$ is an $m \times \ell$ SRFT. Compute QR factorisation of the new coefficient matrix

$$\mathbf{X}^*\mathbf{A} = \mathbf{QRP}^*.$$

Form a preconditioner

$$\mathbf{M} = \mathbf{RP}^*.$$

Solve the preconditioned linear system

$$\big(\mathbf{AM}^{-1}\big)\,\underbrace{(\mathbf{Mx})}_{=:\mathbf{y}} = \mathbf{b}$$

for the new unknown $\mathbf{y}$. Complexity of randomised solver: $O\big((\log(n) + \log(1/\varepsilon))mn + n^3\big)$.

Later improvements include BLENDENPIK by Avron, Maymounkov, Toledo (2010).

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomised Kaczmarz

The classical Kaczmarz algorithm:

With $\mathbf{A} \in \mathbb{R}^{m\times n}$, we seek to solve $\mathbf{Ax} = \mathbf{b}$ through an iterative procedure.

Given an approximate solution $\mathbf{x}_{\rm old}$, compute an improved solution $\mathbf{x}_{\rm new}$ as follows:

(1) Pick a row index $i \in \{1, 2, \dots, m\}$.

(2) Require that $\mathbf{x}_{\rm new}$ is picked so that row i of the system is satisfied exactly.

(3) Within the hyperplane defined by (2), pick $\mathbf{x}_{\rm new}$ as the point closest to $\mathbf{x}_{\rm old}$.

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomised Kaczmarz

The classical Kaczmarz algorithm:

With $\mathbf{A} \in \mathbb{R}^{m \times n}$, we seek to solve $\mathbf{Ax} = \mathbf{b}$ through an iterative procedure.

Given an approximate solution $\mathbf{x}_{\rm old}$, compute an improved solution $\mathbf{x}_{\rm new}$ as follows:

(1) Pick a row index $i \in \{1, 2, \dots, m\}$.

(2) Require that $\mathbf{x}_{\rm new}$ is picked so that row i of the system is satisfied exactly.

(3) Within the hyperplane defined by (2), pick $\mathbf{x}_{\rm new}$ as the point closest to $\mathbf{x}_{\rm old}$.

The resulting formula is $\mathbf{x}_{\rm new} = \mathbf{x}_{\rm old} + \dfrac{\mathbf{b}(i) - \big(\mathbf{A}(i,:) \cdot \mathbf{x}_{\rm old}\big)}{\|\mathbf{A}(i,:)\|^2} \mathbf{A}(i,:)^*$.

Question: How do you pick the row index i in step (1)?

Randomised methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$: Randomised Kaczmarz

The classical Kaczmarz algorithm:

With $\mathbf{A} \in \mathbb{R}^{m \times n}$, we seek to solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ through an iterative procedure.

Given an approximate solution $\mathbf{x}_{\rm old}$, compute an improved solution $\mathbf{x}_{\rm new}$ as follows:

(1) Pick a row index $i \in \{1, 2, \dots, m\}$.

(2) Require that $\mathbf{x}_{\rm new}$ is picked so that row i of the system is satisfied exactly.

(3) Within the hyperplane defined by (2), pick $\mathbf{x}_{\rm new}$ as the point closest to $\mathbf{x}_{\rm old}$.

The resulting formula is $\mathbf{x}_{\rm new} = \mathbf{x}_{\rm old} + \dfrac{\mathbf{b}(i) - \big(\mathbf{A}(i,:) \cdot \mathbf{x}_{\rm old}\big)}{\|\mathbf{A}(i,:)\|^2}\mathbf{A}(i,:)^*$.

Question: How do you pick the row index i in step (1)?

Strohmer & Vershynin (2009): Draw i with probability proportional to $\|\mathbf{A}(i,:)\|$.

Theorem: Let $\mathbf{x}_\star$ denote the exact solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$, and let $\mathbf{x}_k$ denote the k'th iterate of the S&V randomised Kaczmarz method. Then

$$\mathbb{E}\big[\|\mathbf{x}_k - \mathbf{x}_\star\|\big] \leq \left(1 - \frac{1}{\kappa(\mathbf{A})^2}\right)^k \|\mathbf{x}_0 - \mathbf{x}_\star\|,$$

where $\kappa(\mathbf{A})$ is the "scaled" condition number $\kappa(\mathbf{A}) = \|\mathbf{A}\|_{\rm F}\, \|\mathbf{A}^{-1}\|_2$.

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomised Kaczmarz

The classical Kaczmarz algorithm:

With $\mathbf{A} \in \mathbb{R}^{m\times n}$, we seek to solve $\mathbf{Ax} = \mathbf{b}$ through an iterative procedure.

Given an approximate solution $\mathbf{x}_{\rm old}$, compute an improved solution $\mathbf{x}_{\rm new}$ as follows:

(1) Pick a row index $i \in \{1, 2, \dots, m\}$.

(2) Require that $\mathbf{x}_{\rm new}$ is picked so that row i of the system is satisfied exactly.

(3) Within the hyperplane defined by (2), pick $\mathbf{x}_{\rm new}$ as the point closest to $\mathbf{x}_{\rm old}$.

The resulting formula is $\mathbf{x}_{\rm new} = \mathbf{x}_{\rm old} + \dfrac{\mathbf{b}(i) - \big(\mathbf{A}(i,:)\cdot\mathbf{x}_{\rm old}\big)}{\|\mathbf{A}(i,:)\|^2}\mathbf{A}(i,:)^*$.

Question: How do you pick the row index i in step (1)?

Gower & Richtarik (2015): Draw an $m \times \ell$ random map $\mathbf{X}$

$$\mathbf{x}_{\rm new} = \operatorname{argmin}\{\|\mathbf{y} - \mathbf{x}_{\rm old}\| : \ \mathbf{y} \text{ satisfies } \mathbf{X}^*\mathbf{Ay} = \mathbf{X}^*\mathbf{b}\}.$$

Leads to stronger analysis, and a much richer set of dimension reducing maps.

In particular, it improves practical performance since it enables *blocking*.

Note: An ideal weight for a group of rows would be their spanning volume ...

Randomised methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$: Randomised Newton-Schulz

Classical Newton-Schulz for computing $\mathbf{A}^{-1}$: With $\mathbf{A} \in \mathbb{R}^{n\times n}$, we build $\mathbf{B} = \mathbf{A}^{-1}$ through an iterative scheme. Given an approximation $\mathbf{B}_{\rm old}$, the improved one is

$$\mathbf{B}_{\rm new} = \mathbf{B}_{\rm old} - \mathbf{A}\mathbf{B}_{\rm old}\mathbf{A}.$$

Converges rapidly from a good initial guess. But basin of convergence is not large.

Gower & Richtarik (2019): Find $\mathbf{B} = \mathbf{A}^{-1}$ by solving the equation

$$\mathbf{A}^* = \mathbf{A}^*\mathbf{A}\mathbf{B}. \tag{1}$$

Equation (1) is solved through sketching + iteration: Draw an $m \times \ell$ random map $\mathbf{X}$

$$\mathbf{B}_{\rm new} = \text{argmin}\{\|\mathbf{M} - \mathbf{B}_{\rm old}\| : \ \mathbf{M} \text{ satisfies } \mathbf{X}^*\mathbf{A}^* = \mathbf{X}^*\mathbf{A}^*\mathbf{A}\mathbf{M}\}.$$

Equivalent to iteration

$$\mathbf{B}_{\rm new} = \mathbf{B}_{\rm old} - \mathbf{A}^*\mathbf{A}\mathbf{X}\left(\mathbf{X}^*\mathbf{A}^*\mathbf{A}\mathbf{A}^*\mathbf{A}\mathbf{X}\right)^{\dagger}\mathbf{X}^*\mathbf{A}^*\left(\mathbf{A}\mathbf{B}_{\rm old} - \mathbf{I}\right).$$

Detailed error analysis exists. For instance:

The expectation of the error converges exponentially fast, regardless of starting point.

Randomised iterative solvers is a *very* active area: Recent and current work by H. Avron, P. Drineas, L.-H. Lim, M. Mahoney, D. Needell, V. Rokhlin, S. Toledo, J. Tropp, R. Ward, J. Weare, and many more.

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

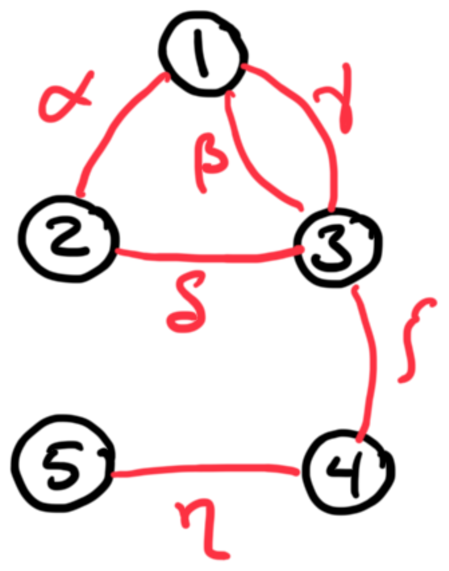
Let us consider a linear system

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

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

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

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



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

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

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

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

Let us consider a linear system

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

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

Standard solution techniques:

- *Multigrid:* Works great for certain classes of matrices.
- *Cholesky:* Compute a decomposition
$$\mathbf{A} = \mathbf{CC}^*,$$
with $\mathbf{C}$ lower triangular. Always works. Numerically stable (when pivoting is used). Can be expensive since the factor $\mathbf{C}$ typically has far more non-zero entries than $\mathbf{A}$.
- *Incomplete Cholesky:* Compute an approximate factorisation
$$\mathbf{A} \approx \mathbf{CC}^*,$$
where $\mathbf{C}$ is constrained to be as sparse as $\mathbf{A}$ (typically the same pattern). Then use CG to solve a system with the preconditioned coefficient matrix $\mathbf{C}^{-1}\mathbf{AC}^{-*}$. Can work very well, hard to analyze.

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

Let us consider a linear system

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

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

Randomised solution techniques:

- *Spielman-Teng (2004):* Complexity $O(m\,\mathrm{poly}(\log n)\ \log(1/\varepsilon))$.

  Relies on graph theoretical constructs (low-stretch trees, graph sparsification, explicit expander graphs, ...). Important theoretical results.

- *Kyng-Lee-Sachdeva-Spielman (2016):* $O(m\,(\log n)^2)$.

  Relies on local sampling only. Much closer to a realistic algorithm.

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

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

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

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

Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: “Rank structured” matrices

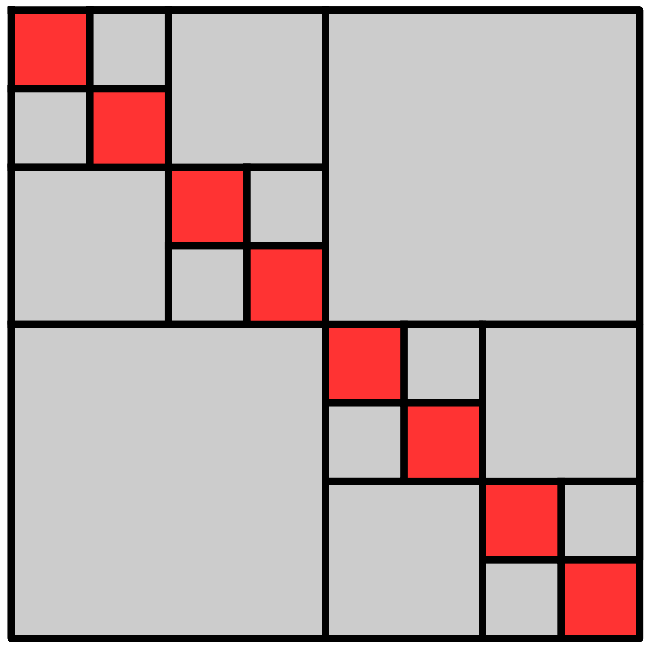
Many matrices in applications have *off-diagonal blocks* that are of low rank:

- Matrices approximating integral equations associated with elliptic PDEs. (Essentially, discretized Calderòn-Zygmund operators.)
- Scattering matrices in acoustic and electro-magnetic scattering.
- Inverses of (sparse) matrices arising upon FEM discretization of elliptic PDEs.
- Buzzwords: $\mathcal{H}$-matrices, HSS-matrices, HBS matrices, …

Using randomised algorithms, we have developed $O(N)$-complexity methods for performing algebraic operations on dense matrices of this type. This leads to:

- *Accelerated direct solvers for elliptic PDEs.*
- *$O(N)$ complexity in many situations.*

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



Randomised methods for solving $\mathbf{Ax} = \mathbf{b}$: “Rank structured” matrices

Let $\mathbf{A}$ be a rank-structured matrix, for which we can rapidly evaluate $\mathbf{x} \mapsto \mathbf{Ax}$ and $\mathbf{x} \mapsto \mathbf{A}^*\mathbf{x}$.

There exist two classes of randomised algorithms for “compressing” $\mathbf{A}$:

Case 1: Suppose that in addition to matvec, we can also evaluate individual entries of $\mathbf{A}$.
Then an HBS (a.k.a. HSS) representation can be computed in $O(N)$ operations.
Very computationally efficient in practice — requires only $O(k)$ matvecs.

- P.G. Martinsson, SIMAX, **32**(4), 2011.
- Later improvements by Jianlin Xia, Sherry Li, etc.

Case 2: If all we have is the matvec, then we can still compute a rank-structured representation of $\mathbf{A}$ using so called “peeling” algorithms. The price we have to pay is that we now need $O(k \times \log N)$ matvecs involving $\mathbf{A}$ and $\mathbf{A}^*$.

The method is still fast in many situations, and does save messy coding work. For instance, without this black-box method, implementing the matrix-matrix multiplication, or changing the partition tree, are quite hard to implement efficiently.

- L. Lin, J. Lu, L. Ying, *Fast construction of hierarchical matrix representation from matrix-vector multiplication*, JCP 2011.
- P.G. Martinsson, SISC, **38**(4), pp. A1959-A1986, 2016.

An example from data science: Kernel ridge regression

The matrices we represent using rank-structured formats are typically *kernel matrices*, which is to say that their entries can be written as

$$\mathbf{A}(i,j) = k(\mathbf{x}_i, \mathbf{x}_j)$$

for some set of points $\{\mathbf{x}_i\}_{i=1}^{n}$ in $\mathbb{R}^d$.

The methods described are designed for problems in scientific computing where the dimension d is moderate. (Say $d < 4$.)

In data science, kernel matrices arise for point sets in much higher dimensions. For such problems, an approach based on *sampling* is often necessary. (“Sketch-to-solve” rather than “sketch-to-precondition”.)

An example from data science: Kernel ridge regression

Task: We are given a set of pairs $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ are data points, and where y_i are corresponding labels. We seek to build a function $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$y_i \approx f(\boldsymbol{x}_i)$$

for every point in the training set. The objective is to predict the label for any new unseen data point $\boldsymbol{x}$.

Methodology: Let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel function that measures how similar a pair of points are, scaled so that

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &\approx 1 \qquad \text{means } \boldsymbol{x} \text{ and } \boldsymbol{y} \text{ are similar,} \\ k(\boldsymbol{x}, \boldsymbol{y}) &\approx 0 \qquad \text{means } \boldsymbol{x} \text{ and } \boldsymbol{y} \text{ are uncorrelated.} \end{aligned}$$

It is the job of the modeler to provide a “good” kernel function.

We then approximate f using the formula $f(\boldsymbol{x}) = \sum_{i=1}^n k(\boldsymbol{x}, \boldsymbol{x}_i)\, \alpha_i$, where the *weights* $\{\alpha_i\}_{i=1}^n$ are computed using the formula $\boldsymbol{\alpha} = \big(\mathbf{K} + \lambda n \mathbf{I}\big)^{-1}\mathbf{y}$, where $\mathbf{K}$ is the $n \times n$ matrix with entries $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The number λ is a regularization parameter.

Challenge: $\mathbf{K}$ is very large, and computing an individual entry can be expensive.

Randomized solution: Draw an in index vector $\mathbf{J} \subset \{1, 2, \dots, n\}$ holding k sampled indices, and replace $\mathbf{K}$ by the formula

$$\mathbf{K}_{\rm approx} = \mathbf{K}(:\,, J)\, \mathbf{K}(J, J)^{\dagger}\, \mathbf{K}(J, :\,).$$

Key points:

- Randomised low-rank approximation ("randomised SVD").
  - Superior performance in many regards, in particular for very large problems.
  - For a fixed number of matrix-vector multiplies, Krylov methods are more accurate.
- Essential benefit of randomisation in linear algebra: *Reduces communication.*
  - Enables processing of huge data sets. (Out-of-core / streaming / cloud computing / . . .)
  - Very fast on GPUs, distributed memory machines, etc.
- There is exciting ongoing work on randomised methods for solving $\mathbf{Ax} = \mathbf{b}$.
  - Acceleration of existing $O(n^3)$ solvers — work very well, recommended without caveats.
  - Randomized preconditioners — currently work very well in some environments.
  - Two quite different methodologies:

    Sketch-to-precondition: Safe, highly recommended.

    Sketch-to-solve: Enables solvers for otherwise inaccessible problems.
  - Rank structured matrices — promising, but still work in progress.
- Even though the algorithms are randomised, *the output can be trusted.*

  The probability of failure can be made *extremely* low (say 10^{-10}).

  In most situations, you can explicitly compute the residual error.

  Cf. Monte Carlo vs. Las Vegas methods.

Future and ongoing work:

Postdoc position available!

1. *Accelerate full factorisations of matrices.*

   New randomised column pivoted QR algorithm is much faster than LAPACK.

   New "UTV" factorisation method is almost as accurate as SVD and much faster.

2. *Randomised algorithms for structured matrices.*

   Use randomisation to accelerate key numerical solvers for PDEs, for simulating Gaussian processes, etc.

3. *[High risk/high reward] Accelerate linear solvers for "general" systems* $\mathbf{Ax} = \mathbf{b}$*.*

   The goal is methods with complexity $O(n^\gamma)$ for $\gamma < 3$. Crucially, we seek methods that retain stability, and have high practical efficiency for realistic problem sizes.

4. *Use randomised projections to accelerate non-linear algebraic tasks.*

   Faster nearest neighbor search, faster clustering algorithms, etc. The idea is to use randomised projections for *sketching* to develop a rough map of a large data set. Then use high-accuracy deterministic methods for the actual computation.

Future and ongoing work:

Postdoc position available!

1. *Accelerate full factorisations of matrices.*

   New randomised column pivoted QR algorithm is much faster than LAPACK.

   New "UTV" factorisation method is almost as accurate as SVD and much faster.

2. *Randomised algorithms for structured matrices.*

   Use randomisation to accelerate key numerical solvers for PDEs, for simulating Gaussian processes, etc.

3. *[High risk/high reward] Accelerate linear solvers for "general" systems* $\mathbf{Ax} = \mathbf{b}$.

   The goal is methods with complexity $O(n^\gamma)$ for $\gamma < 3$. Crucially, we seek methods that retain stability, and have high practical efficiency for realistic problem sizes.

4. *Use randomised projections to accelerate non-linear algebraic tasks.*

   Faster nearest neighbor search, faster clustering algorithms, etc. The idea is to use randomised projections for *sketching* to develop a rough map of a large data set. Then use high-accuracy deterministic methods for the actual computation.

☞ Great potential for new discoveries in linear algebra!

Papers (see also `http://users.oden.utexas.edu/~pgm/main_publications.html`):

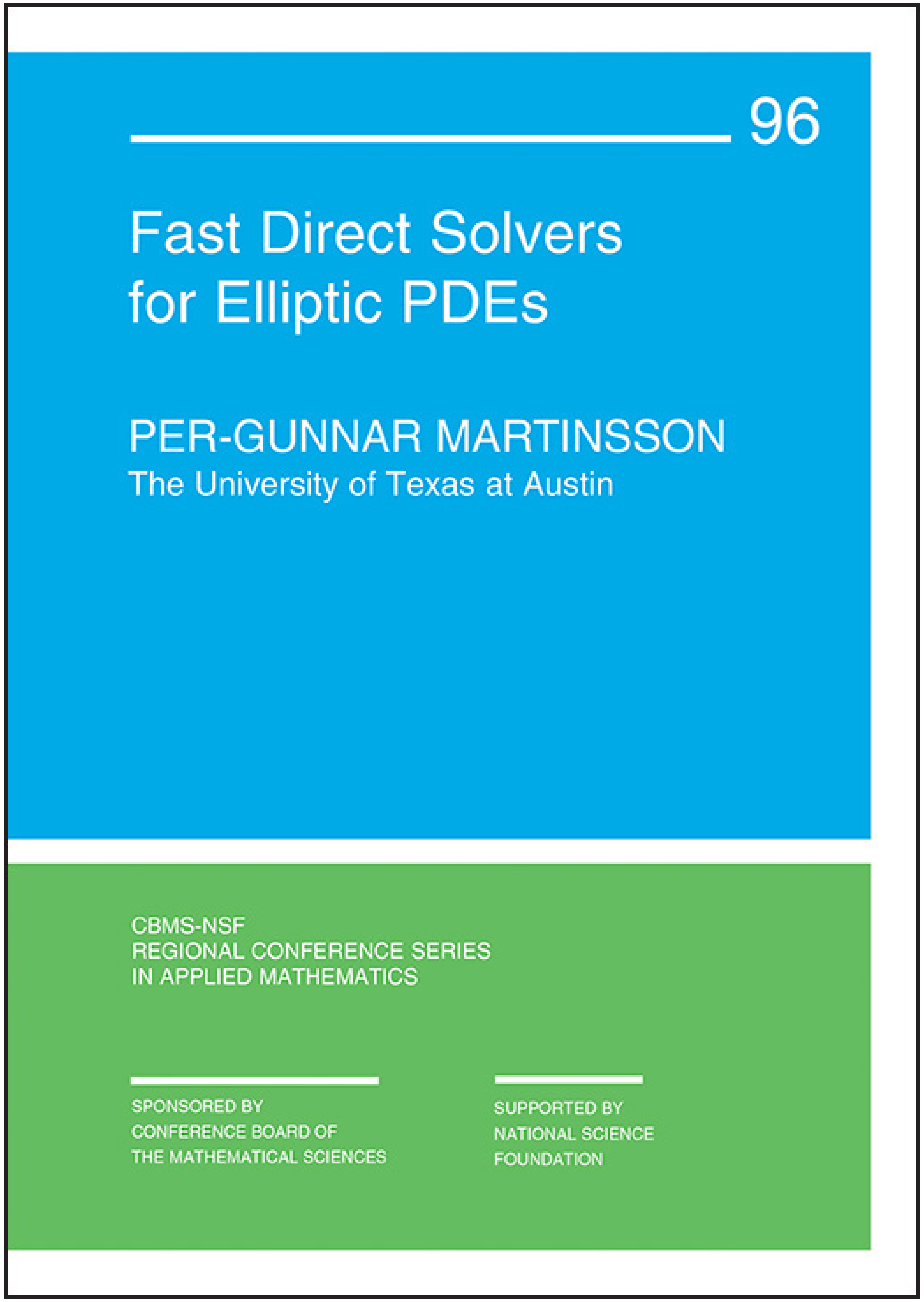
- P.G. Martinsson, J. Tropp, " Randomized Numerical Linear Algebra: Foundations & Algorithms." *Acta Numerica,* 2020. *Available now as arxiv:2002.01387*
- P.G. Martinsson, "Fast Direct Solvers for Elliptic PDEs." SIAM/CBMS, Dec. 2019.
- P.G. Martinsson, "Randomized Methods for Matrix Computations." In the 2018 book *The Mathematics of Data,* published by AMS. See also arxiv.org #1607.01649.
- N. Halko, P.G. Martinsson, J. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." *SIAM Review*, 2011.
- E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, and M. Tygert, "Randomized algorithms for the low-rank approximation of matrices". *PNAS*, **104**(51), 2007.

Tutorials, summer schools, etc:

- 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 packages:

- Column pivoted QR: `https://github.com/flame/hqrrp` (much faster than LAPACK!)
- Randomized UTV: `https://github.com/flame/randutv`
- RSVDPACK: `https://github.com/sergeyvoronin`
- ID: `http://tygert.com/software.html`



96

Fast Direct Solvers for Elliptic PDEs

PER-GUNNAR MARTINSSON
The University of Texas at Austin

CBMS-NSF
REGIONAL CONFERENCE SERIES
IN APPLIED MATHEMATICS

SPONSORED BY
CONFERENCE BOARD OF
THE MATHEMATICAL SCIENCES

SUPPORTED BY
NATIONAL SCIENCE
FOUNDATION