Randomized algorithms for large scale linear algebra

Per-Gunnar Martinsson

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

University of Texas at Austin

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

Slides: `http://users.ices.utexas.edu/∼pgm/main_talks.html`

Research support by:

Randomized dimension reduction

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

- Suppose the points almost live on a linear subspace of (small) dimension k.

  Find a basis for the "best" subspace.
- Given k, find the subset of k vectors with maximal spanning volume.
- Suppose the points almost live on a low-dimensional nonlinear manifold.

  Find a parameterization of the manifold.
- Given k, find for each vector $\mathbf{a}^{(j)}$ its k closest neighbors.
- Partition the points into clusters.

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

Idea: Find an embedding $f : \mathbb{R}^m \to \mathbb{R}^d$ for $d \ll m$ that is almost isometric in the sense

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

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

Randomized dimension reduction

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

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

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

Idea: Find an embedding $f\,:\,\mathbb{R}^m \to \mathbb{R}^d$ for $d \ll m$ that is almost isometric in the sense

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

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

Lemma [Johnson-Lindenstrauss]: For $d \sim \log(n)$, there exists a well-behaved embedding f that "approximately" preserves distances.

To be precise, we have:

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

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

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

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

You can prove that if you pick d as specified, and draw a Gaussian random matrix **R** of size $d \times m$, then there is a positive likelihood that the map

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

satisfies the criteria.

Practical problem: You have two bad choices:

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

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

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

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

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

(A) *Randomized sketching:*

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

(B) *Deterministic post-processing:*

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

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

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

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

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

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

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

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

Jones, Osipov, Rokhlin, 2011

Outline of talk

(1) Randomized dimension reduction — introduction.

(2) Computing low rank approximations (“randomized singular value decomposition”).

(3) Solving $\mathbf{Ax} = \mathbf{b}$.

Outline of talk

(1) Randomized dimension reduction — introduction.

(2) Computing low rank approximations (“randomized singular value decomposition”).

(3) Solving $\mathbf{Ax} = \mathbf{b}$.

Randomized SVD:

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

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

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

Randomized SVD:

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

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

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

(A) *Randomized sketching:*

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

(B) *Deterministic post-processing:*

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

Randomized SVD:

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

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

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

(A) *Randomized sketching:*

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 $\mathrm{ran}(\mathbf{Q}) = \mathrm{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

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

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

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

Randomized SVD:

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

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

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

(A) *Randomized sketching:*

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)`

(B) *Deterministic post-processing:*

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

B.2 Form SVD of the matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, 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 $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.

Randomized SVD:

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

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

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

(A) *Randomized sketching:*

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)`

(B) *Deterministic post-processing:*

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

B.2 Form SVD of the matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, 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 $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.

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

Randomized SVD:

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

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

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

(A) *Randomized sketching:*

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 $\mathrm{ran}(\mathbf{Q}) = \mathrm{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

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

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

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

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

Randomized SVD:

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

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

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

(A) *Randomized sketching:*

A.1 Draw an $n \times k$ Gaussian random matrix **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 **Q** such that $\text{ran}(\mathbf{Q}) = \text{ran}(\mathbf{Y})$. `[Q, ~] = qr(Y)`

(B) *Deterministic post-processing:*

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

B.2 Form SVD of the matrix **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 randomized projections are fine, since all we need is a subspace that captures “the essential” part of the range. Pollution from unwanted singular modes is harmless, as long as we capture the dominant ones. By drawing p extra samples (for, say, $p = 5$ or $p = 10$), we make the risk of missing anything important essentially zero.

Randomized SVD:

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

Randomized SVD:

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

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

(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 $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.

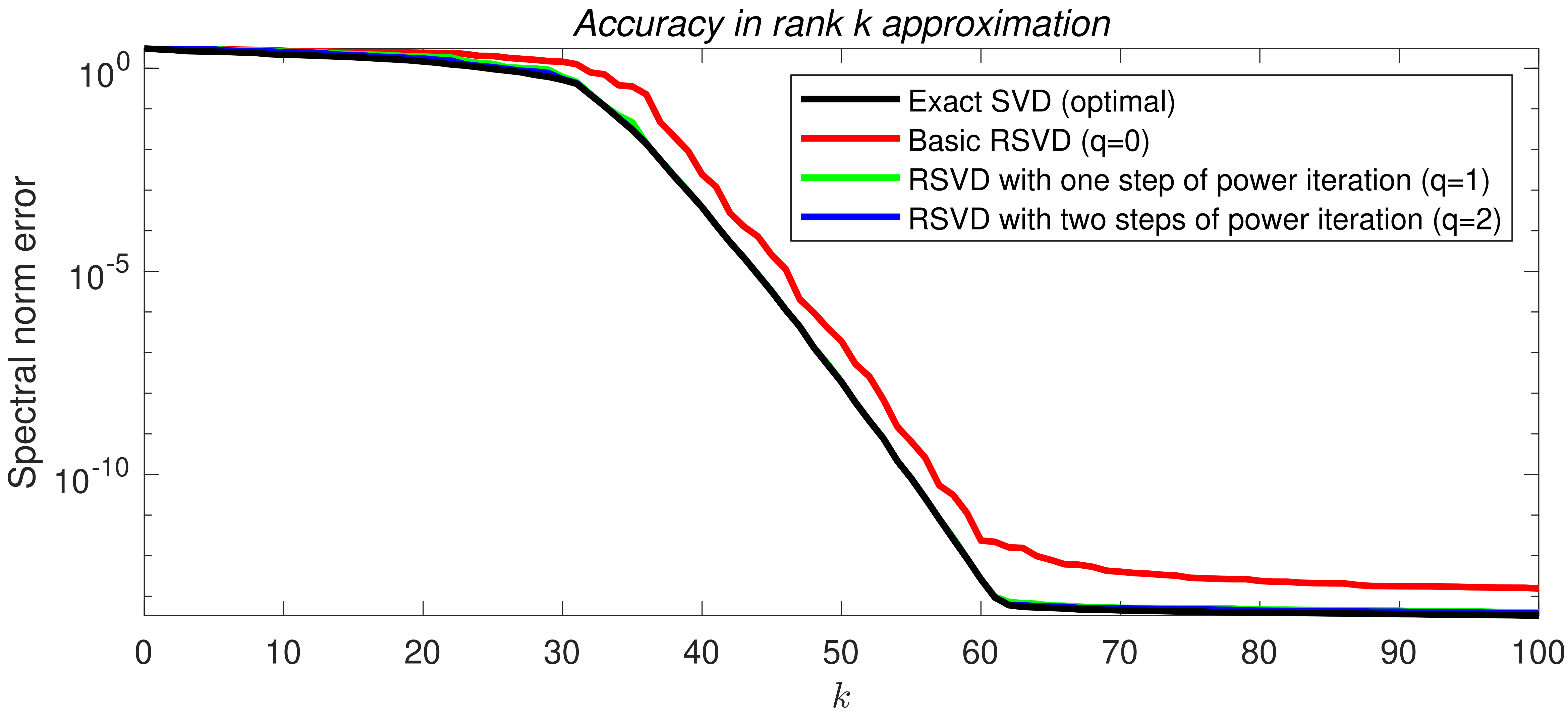
(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

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

- Analogous schemes exist for computing "structure preserving" factorizations where a number of the columns/rows are chosen to serve as a basis for the column/row space. "Interpolative decomposition" / "CUR decomposition" / "skeletonization" / ...
  → *Relaxed solution to "maximal spanning volume" problem on first slide.*

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

Randomized SVD:



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

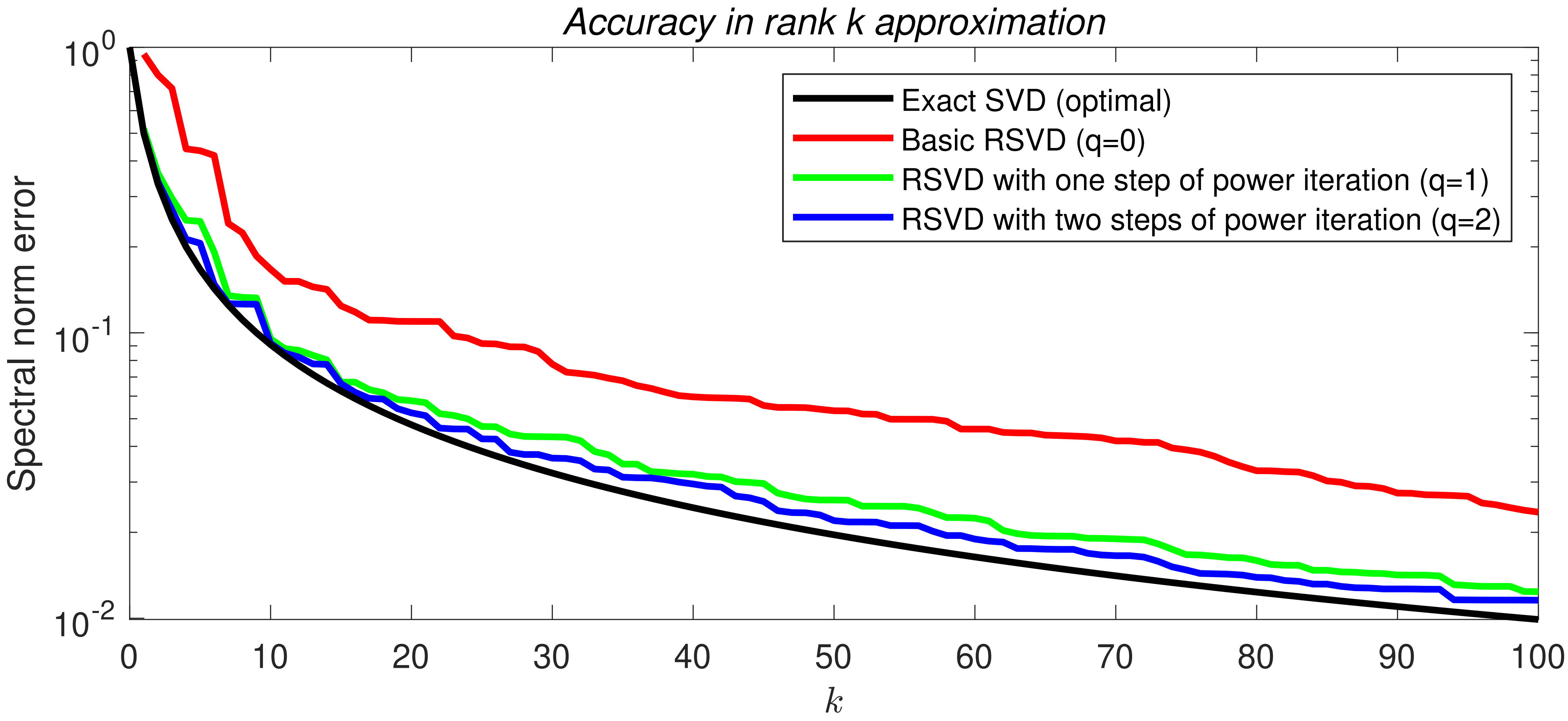
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

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

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

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.

Randomized SVD:



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

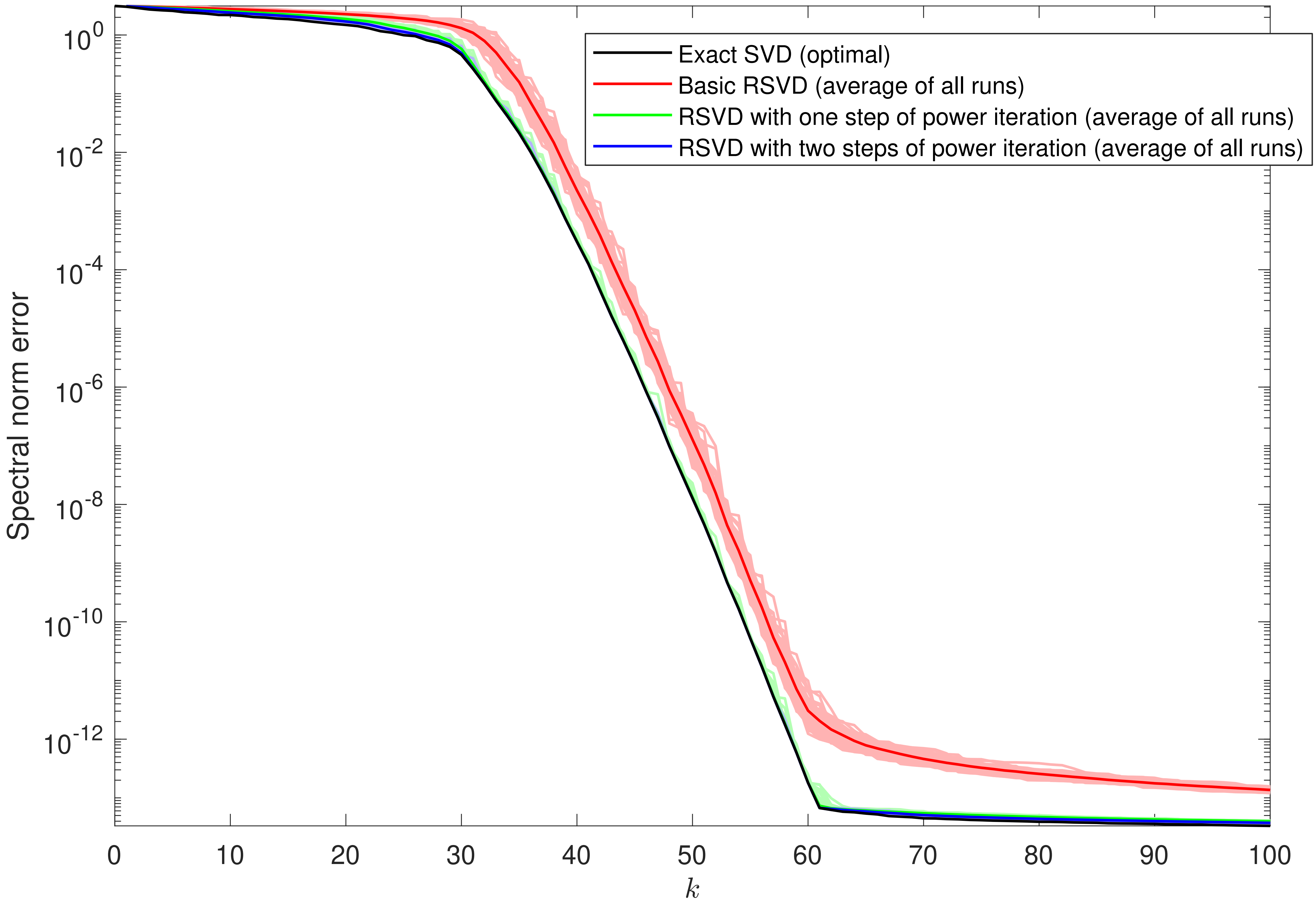
$$e_k = \|\mathbf{A} - \mathbf{P}_k\mathbf{A}\|,$$

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

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

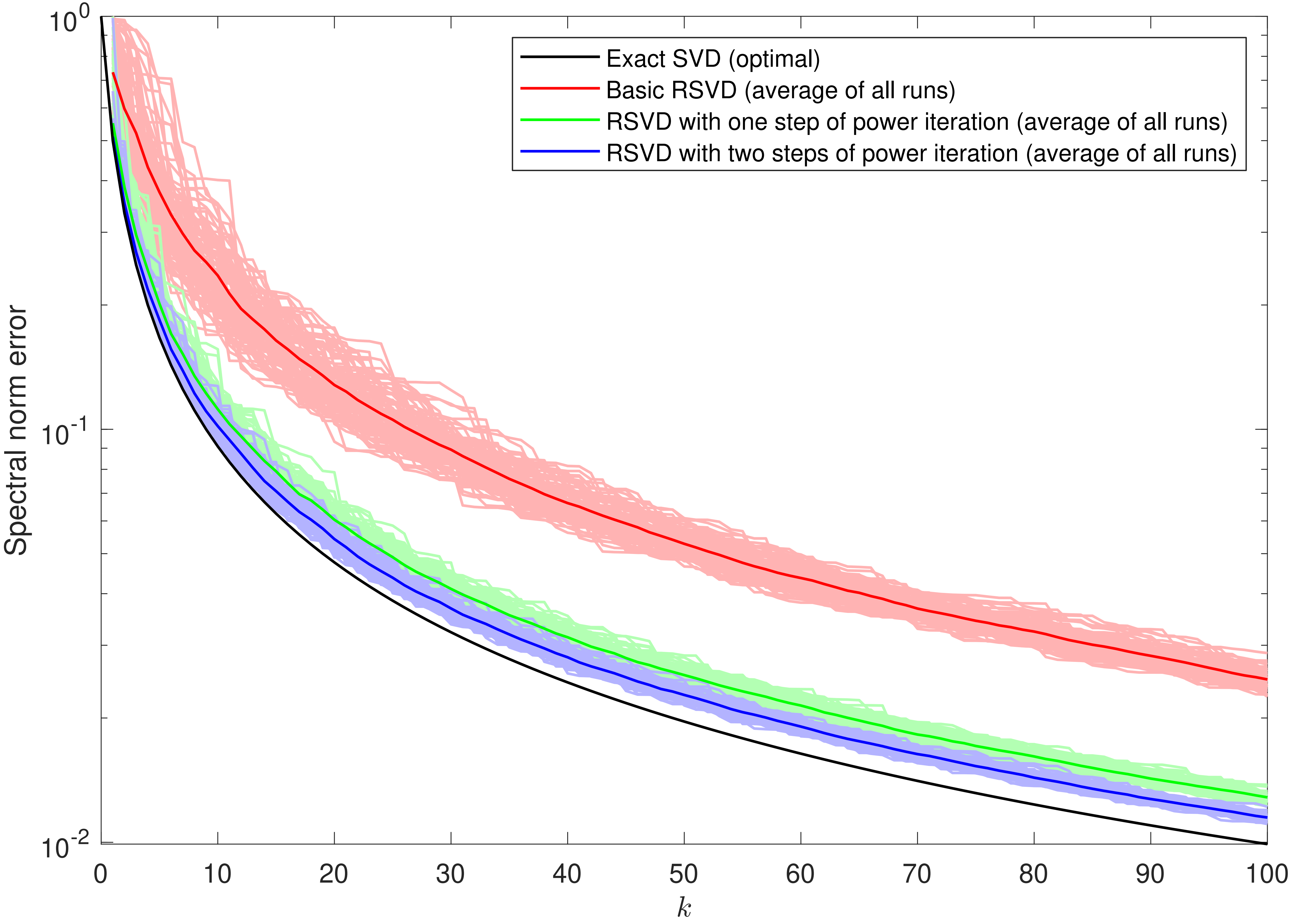
and where $\mathbf{G}$ is a Gaussian random matrix. (Recall that $\mathbf{P}_k\mathbf{A} = \mathbf{U}_k\mathbf{D}_k\mathbf{V}_k^*$.)
The matrix $\mathbf{A}$ now has singular values that decay slowly.

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



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

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



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

Randomized SVD:

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

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

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

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

Randomized SVD:

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

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

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

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

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

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

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

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

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

Theorem: Let $\mathbf{A}$ be an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Let k be a target rank, and let p be an over-sampling parameter such that $p \geq 2$ and $k + p \leq \min(m,n)$. Let $\mathbf{R}$ be a Gaussian random matrix of size $n \times (k+p)$ and set $\mathbf{Q} = \texttt{orth}(\mathbf{AR})$. Then the average error satisfies

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|_{\rm Fro}\big] \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2},$$

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

Randomized SVD:

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, . . .

Randomized SVD:

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

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

(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

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

- $\mathbf{D}$ 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, $\mathbf{F}_{pq} = \frac{1}{\sqrt{n}}\,e^{-2\pi i(p-1)(q-1)/n}$.
- $\mathbf{S}$ 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 $\mathbf{D}\,\mathbf{F}$.)

Randomized SVD:

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

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

(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, …

- 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 …

Randomized SVD:

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

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

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

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

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

Outline of talk

(1) Randomized dimension reduction — introduction.

(2) Computing low rank approximations (“randomized singular value decomposition”).

(3) Solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n\times n}$. (Focus is on the case where **A** is dense.)

Outline of talk

(1) Randomized dimension reduction — introduction.

(2) Computing low rank approximations (“randomized singular value decomposition”).

(3) Solving $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n\times n}$. (Focus is on the case where **A** is dense.)

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

Outline of talk

(1) Randomized dimension reduction — introduction.

(2) Computing low rank approximations (“randomized singular value decomposition”).

(3) Solving $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n \times n}$. (Focus is on the case where **A** is dense.)

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

Observation: When **A** is well-conditioned, iterative methods converge rapidly.
Worst case complexity for solving $\mathbf{Ax} = \mathbf{b}$ to precision ε is then $O(n^2 \log(1/\varepsilon))$.
The challenge concerns matrices that are ill-conditioned.
(Or, to be more precise, whose spectra are not clustered.)

Randomized 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 factorization 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

Randomized 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 factorization 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.

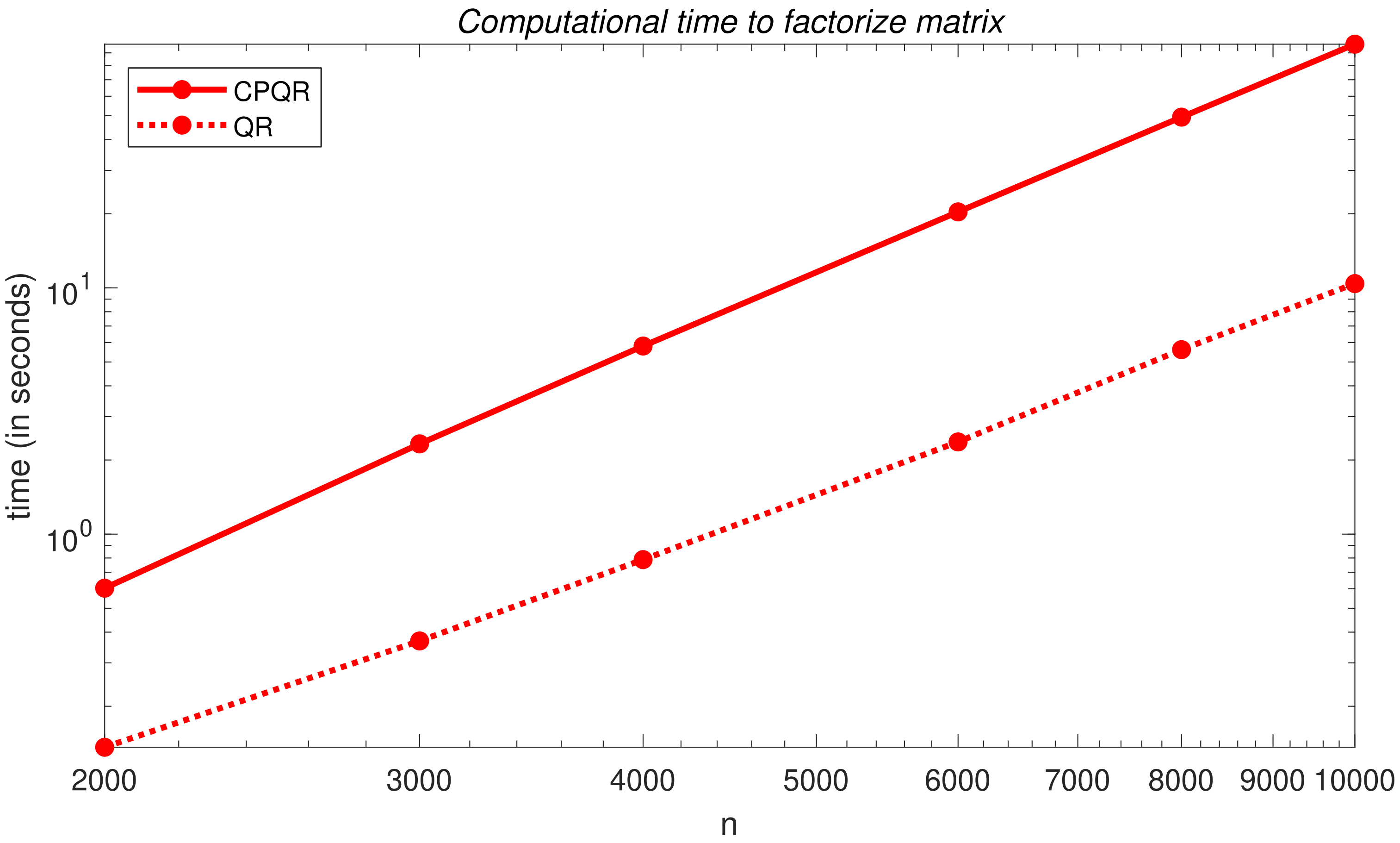
The “robust” factorizations to the right all depend on algorithms that proceed through a sequence of rank-one updates to the matrix. This makes them slow when executed on modern hardware (even on a single core).

Randomized methods for solving **Ax** $=$ **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 factorization 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.



Randomized 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.

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

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

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

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

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

(2) Perform unpivoted QR on the new matrix

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

The resulting factorization

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

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

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

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

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

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

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

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

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

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

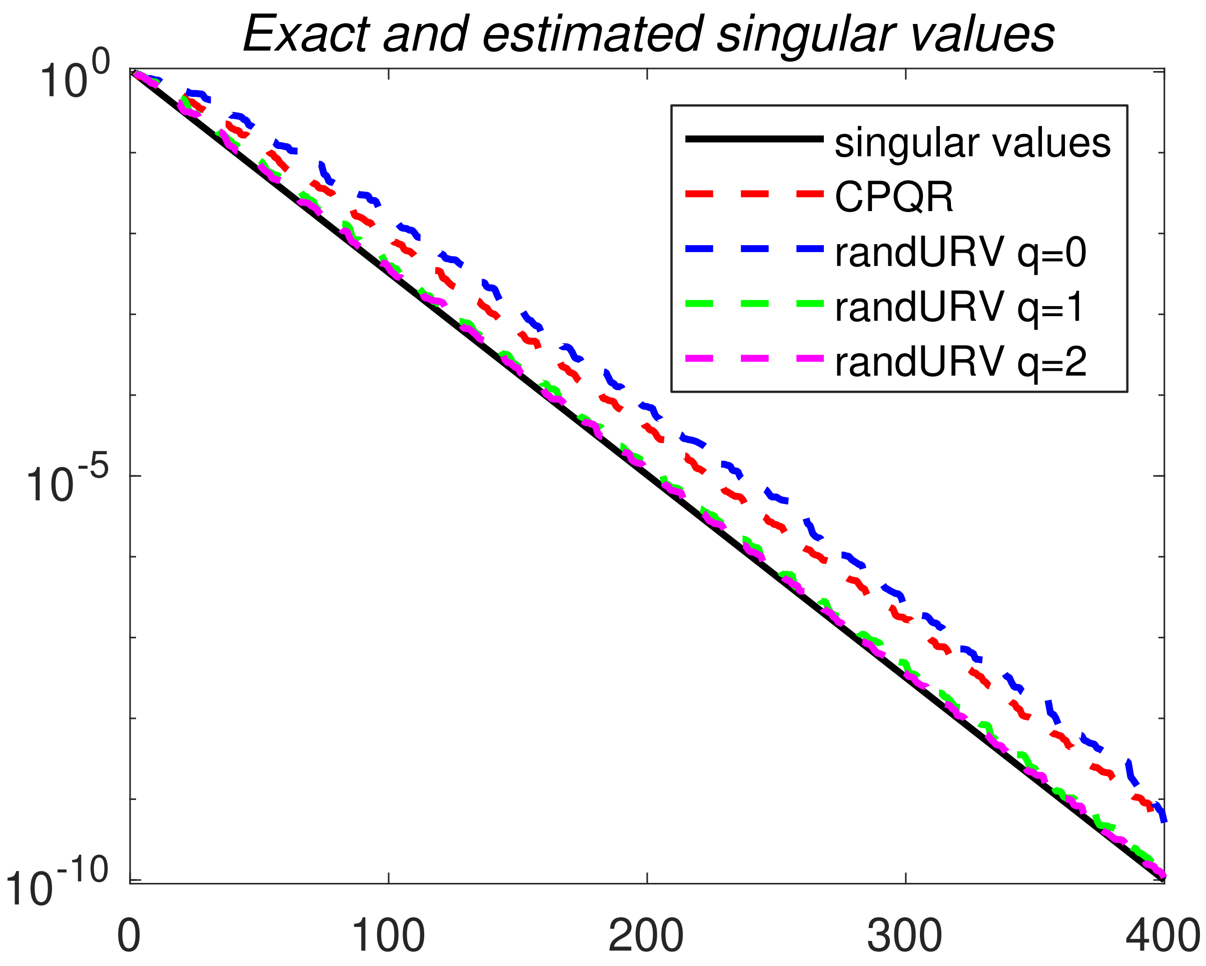
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

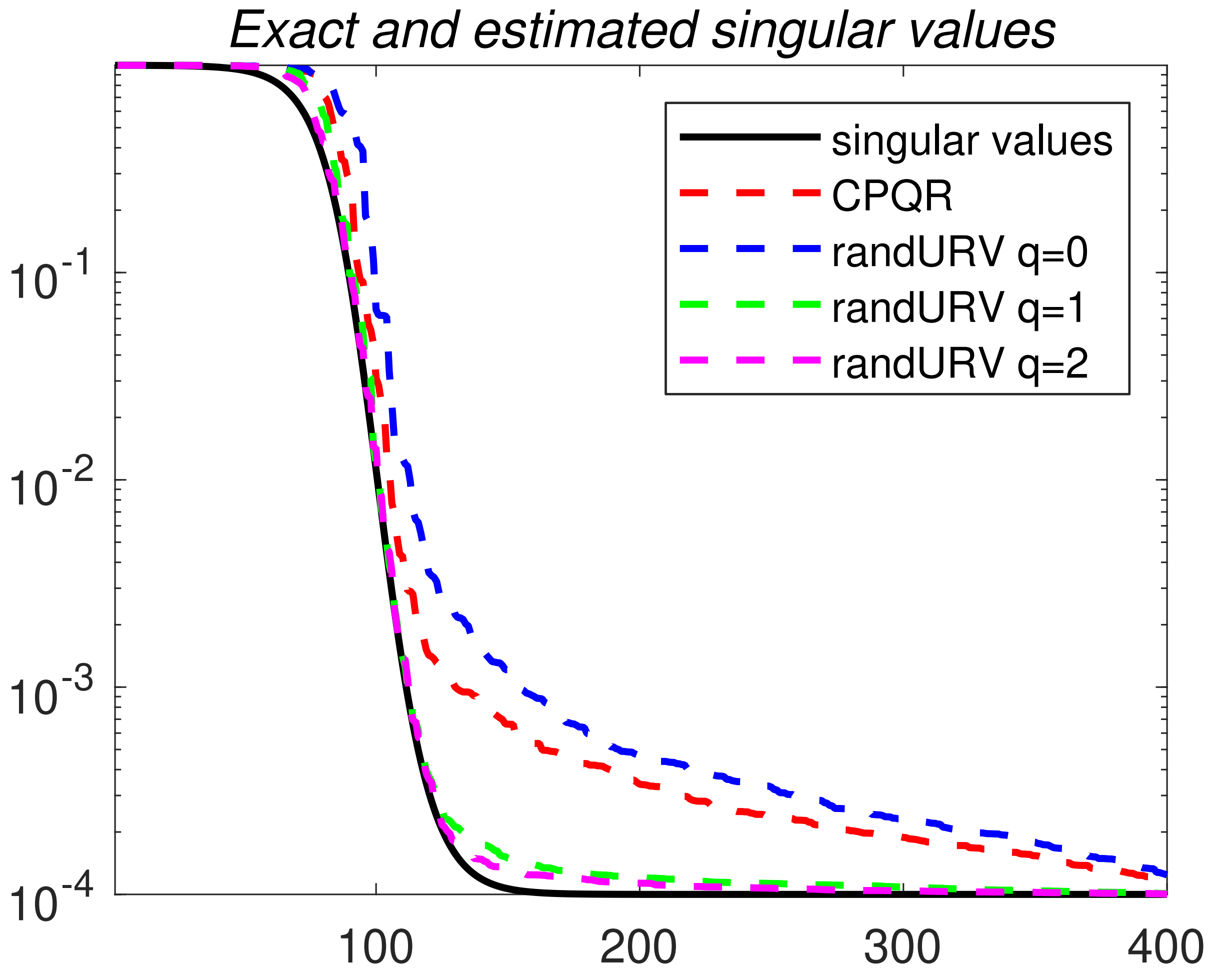
1. Draw a Gaussian random matrix $\mathbf{G}$ and form $\mathbf{Y} = (\mathbf{AA}^*)^q\mathbf{G}$.
2. Perform unpivoted QR on $\mathbf{Y}$ so that $\mathbf{Y} = \mathbf{VR}_{\rm trash}$.
3. Perform unpivoted QR on $\mathbf{AV}$ so that $\mathbf{AV} = \mathbf{UR}$.

This results in a factorization

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

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





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

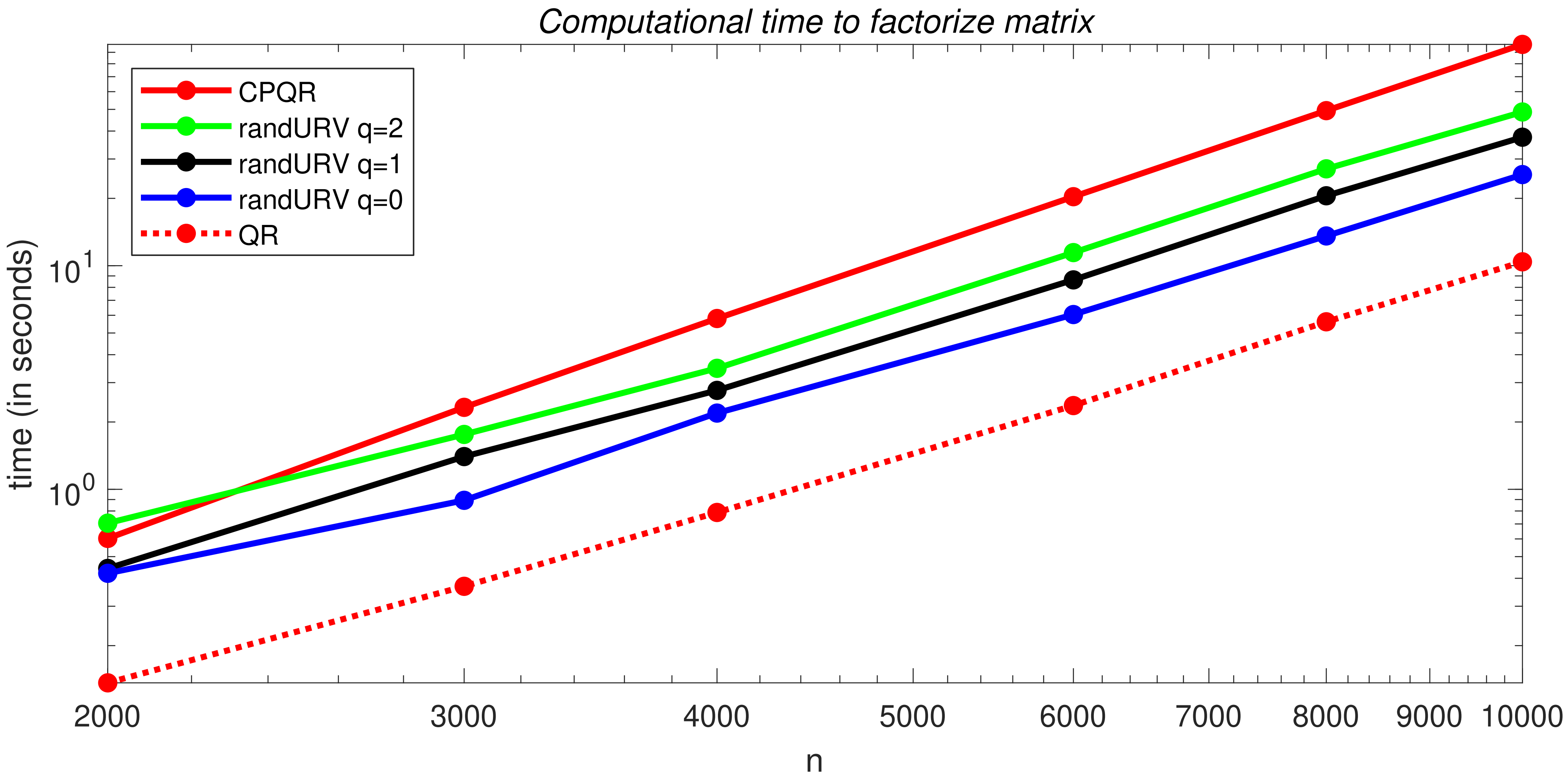
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

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

This results in a factorization

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

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



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

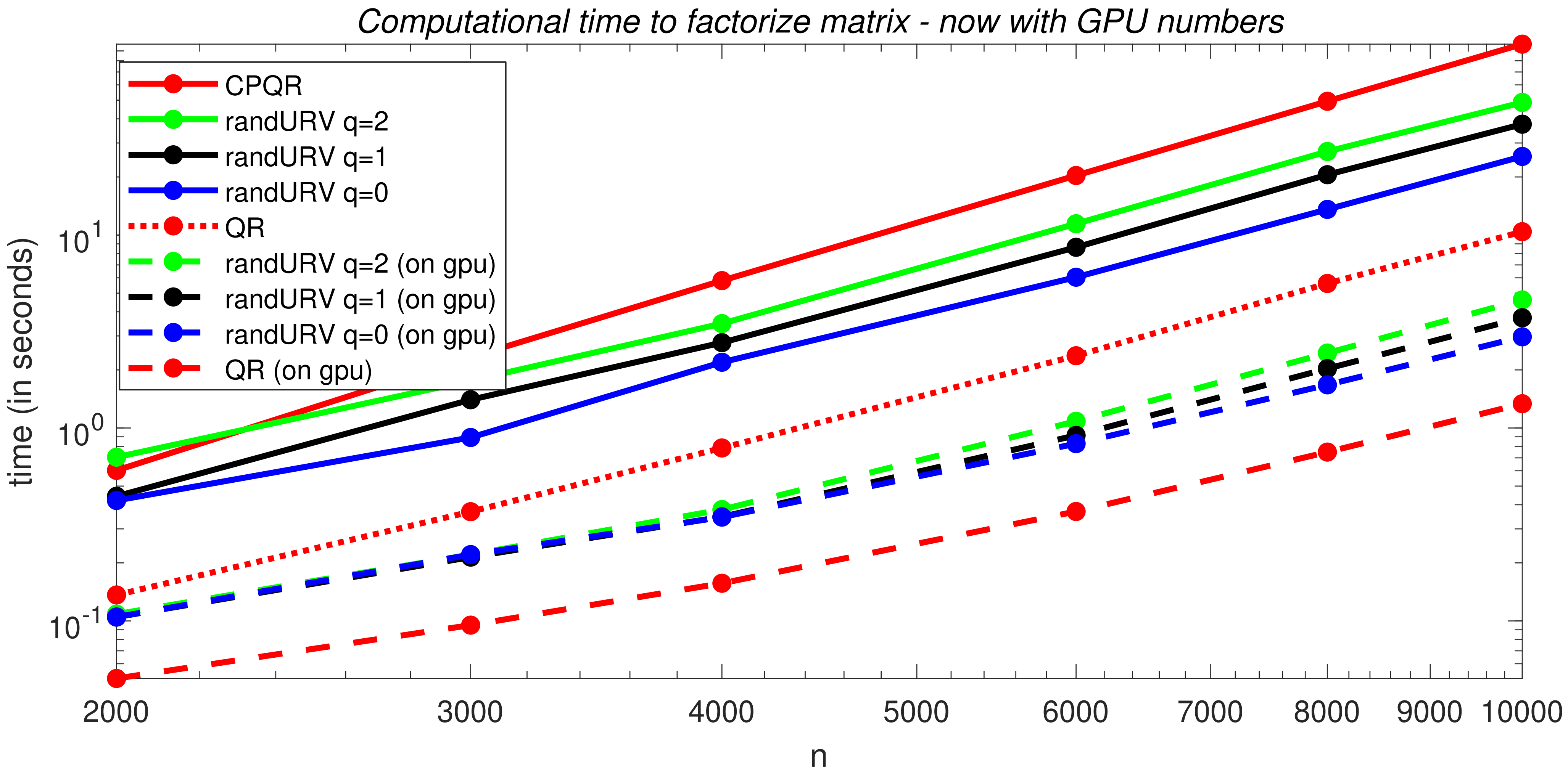
Improved URV factorization: Do q steps of power iteration (for $q = 1$ or $q = 2$, say):

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

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

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



Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$: $O(n^3)$ complexity methods

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

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

This results in a factorization

$$\mathbf{A} = \big(\mathbf{A}\mathbf{V}\big)\mathbf{V}^* = \mathbf{U}\mathbf{R}\mathbf{V}^*$$

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);
```

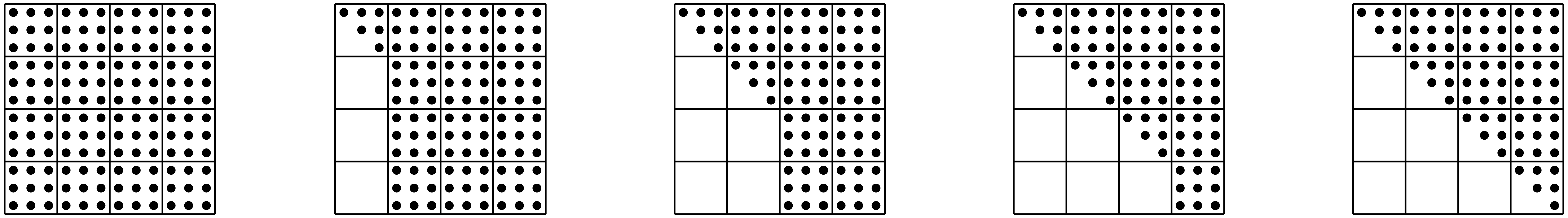
Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ complexity methods

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

$$\begin{array}{cccccc} \mathbf{A} & \mathbf{P} & \approx & \mathbf{Q} & \mathbf{R}, \\ n \times n & n \times n & & n \times n & n \times n \end{array}$$

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 randomized sampling.

Each $\mathbf{Q}_j$ is a product of Householder reflectors.

The key challenge has been to find good permutation matrices.

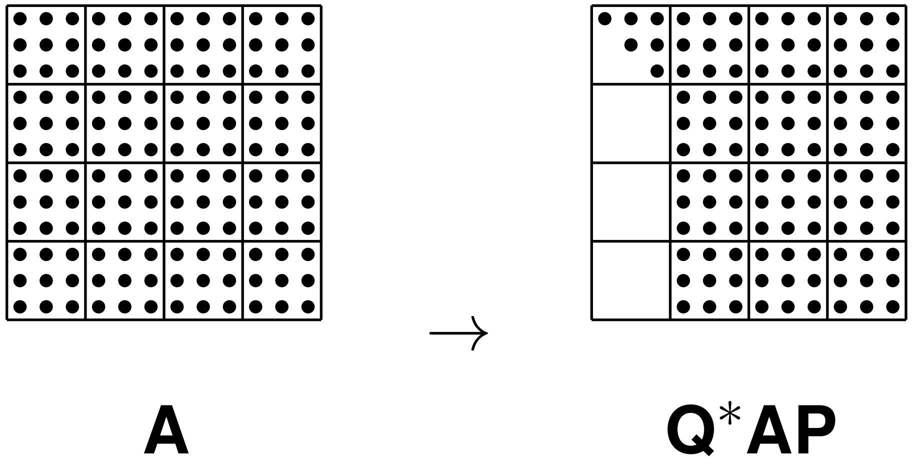
We seek $\mathbf{P}_j$ so that the set of b chosen columns *has maximal spanning volume.*

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

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

How to do block pivoting using randomization:

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

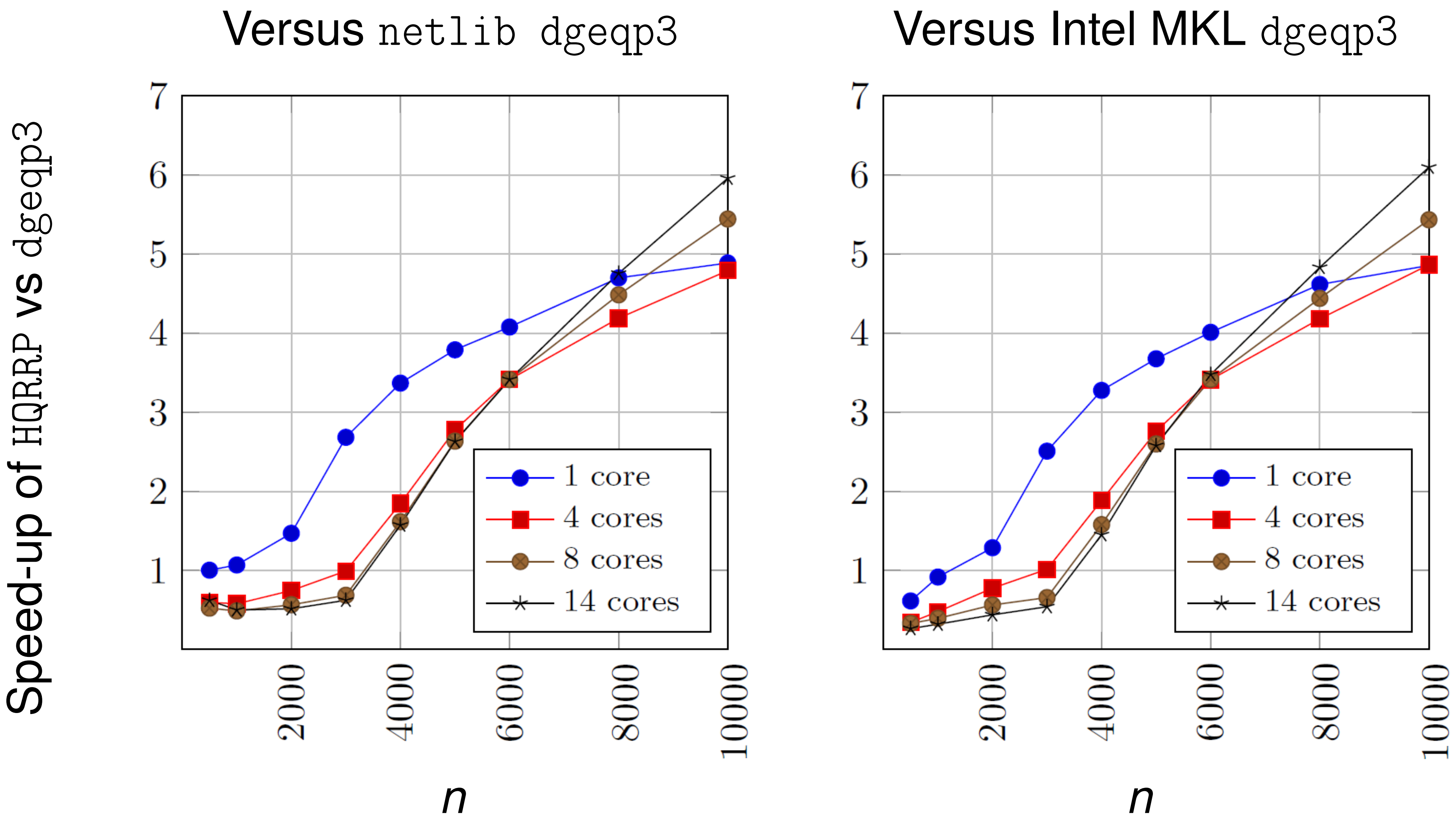
$$\begin{array}{cccc} \mathbf{Y} & = & \mathbf{G} & \mathbf{A} \\ b \times n & & b \times m & m \times n \end{array}$$

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**:

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

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


Versus `netlib dgeqp3`

Versus Intel MKL `dgeqp3`

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/`

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

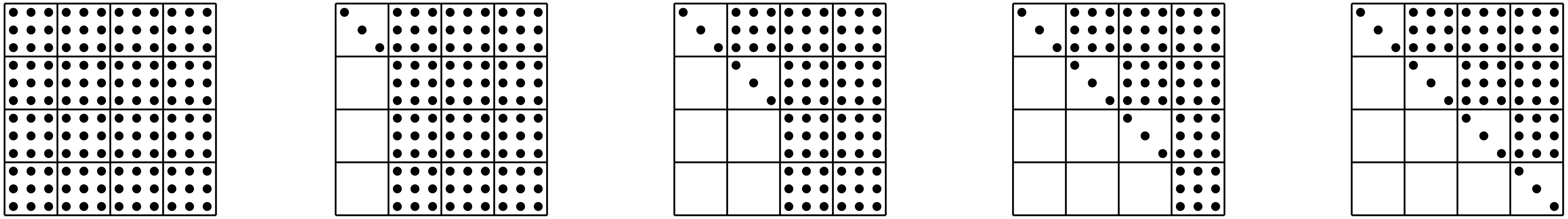
Given a dense $n \times n$ matrix $\mathbf{A}$, compute a factorization

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

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.

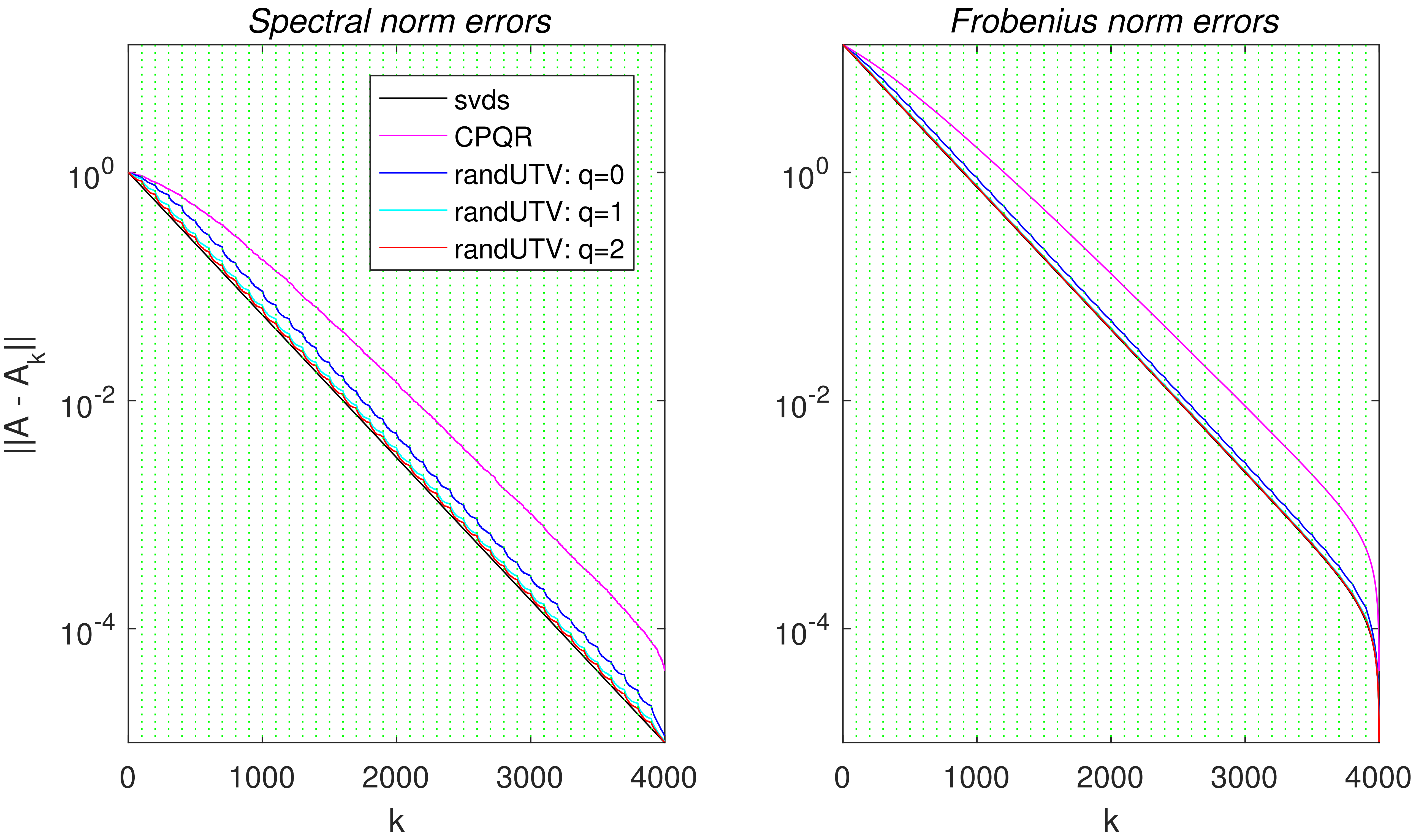
The technique proposed is based on a blocked version of classical Householder QR:



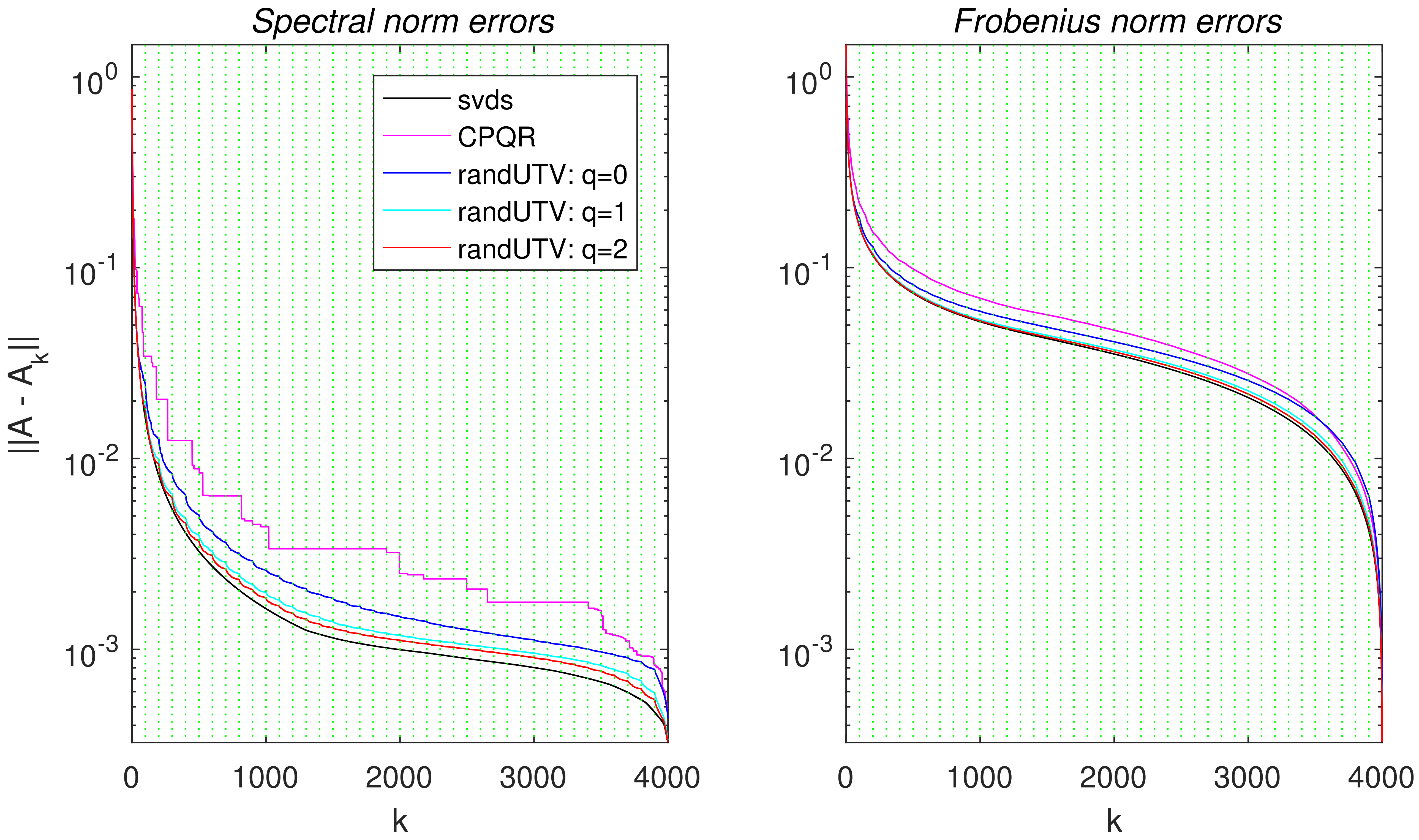
$\mathbf{A}_0 = \mathbf{A}$ $\quad$ $\mathbf{A}_1 = \mathbf{U}_1^*\mathbf{A}_0\mathbf{V}_1$ $\quad$ $\mathbf{A}_2 = \mathbf{U}_2^*\mathbf{A}_1\mathbf{V}_2$ $\quad$ $\mathbf{A}_3 = \mathbf{U}_3^*\mathbf{A}_2\mathbf{V}_3$ $\quad$ $\mathbf{A}_4 = \mathbf{U}_4^*\mathbf{A}_3\mathbf{V}_4$

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

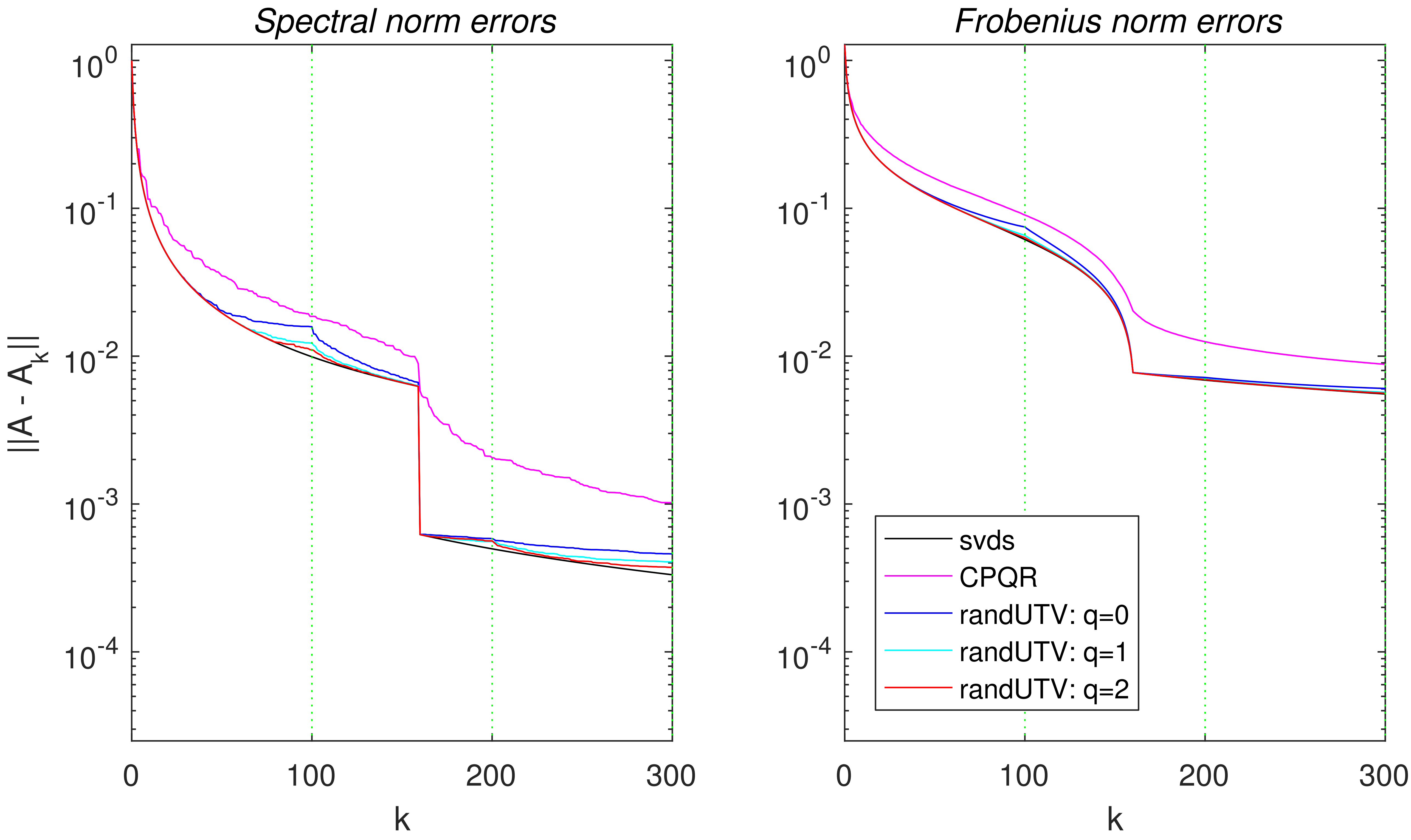
Our objective is in each step to find an approximation *to the linear subspace* spanned by the b dominant singular vectors of a matrix. The randomized range finder is perfect for this, especially when a small number of power iterations are performed. Easier and more natural than choosing pivoting vectors.



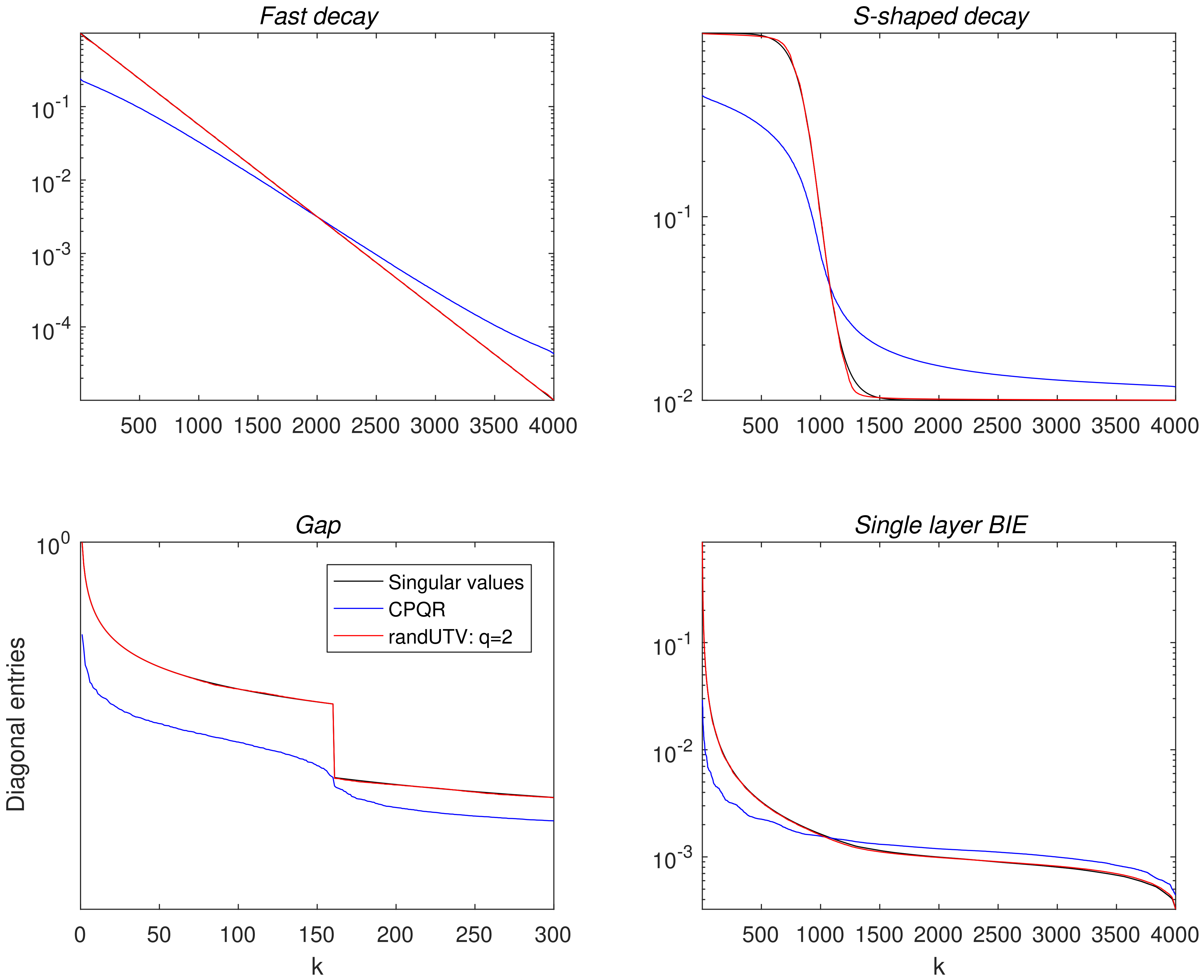
Rank-k approximation errors for the matrix "Fast Decay" of size 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*



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



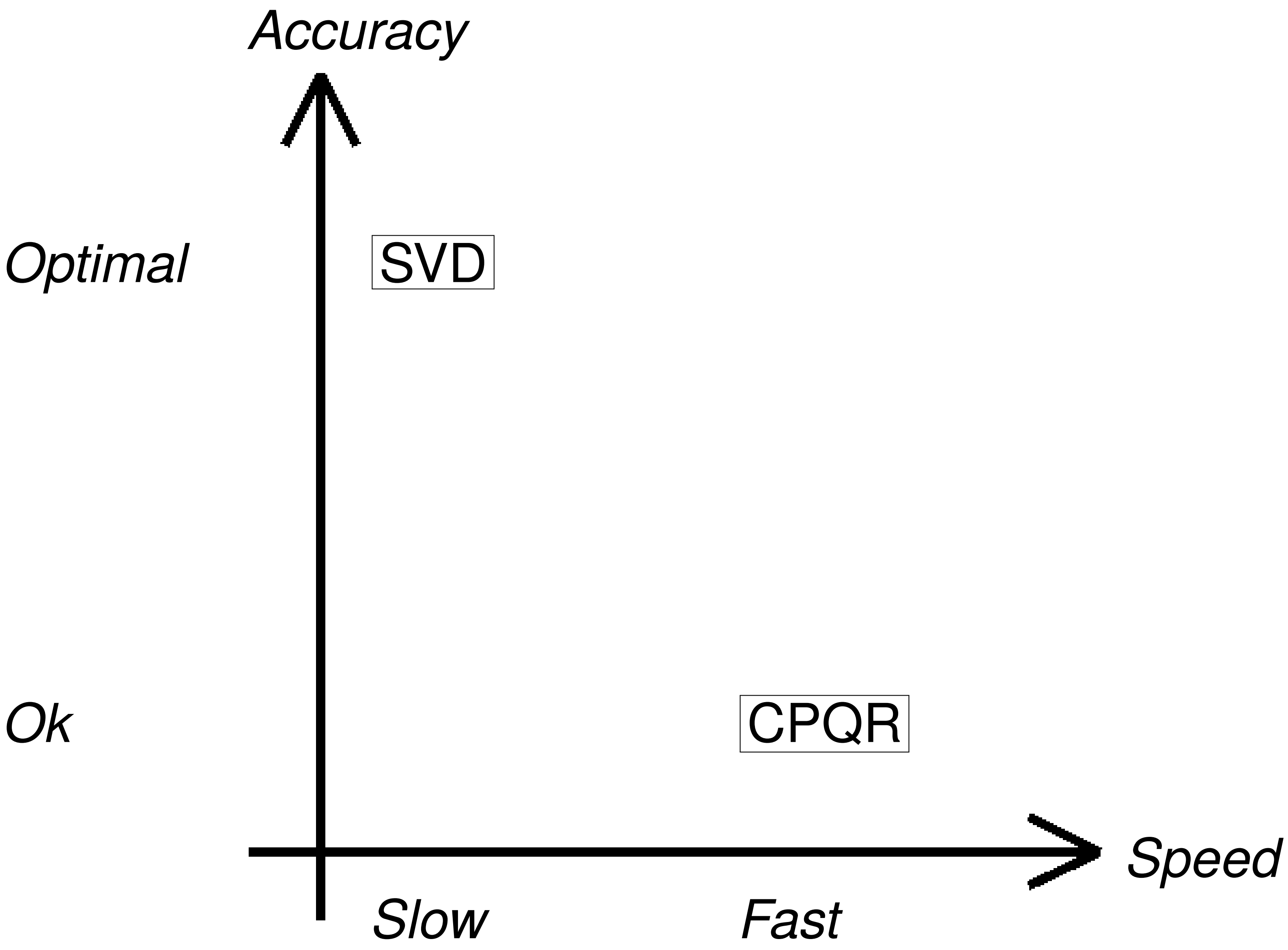
Rank-k approximation errors for k $\leq$ *300 for the matrix "Gap" of size* 4000×4000*. The black lines mark the theoretically minimal errors. The block size was* $b = 100$ *and the green vertical lines mark block limits.*



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

Randomized 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:



Accuracy

Optimal SVD

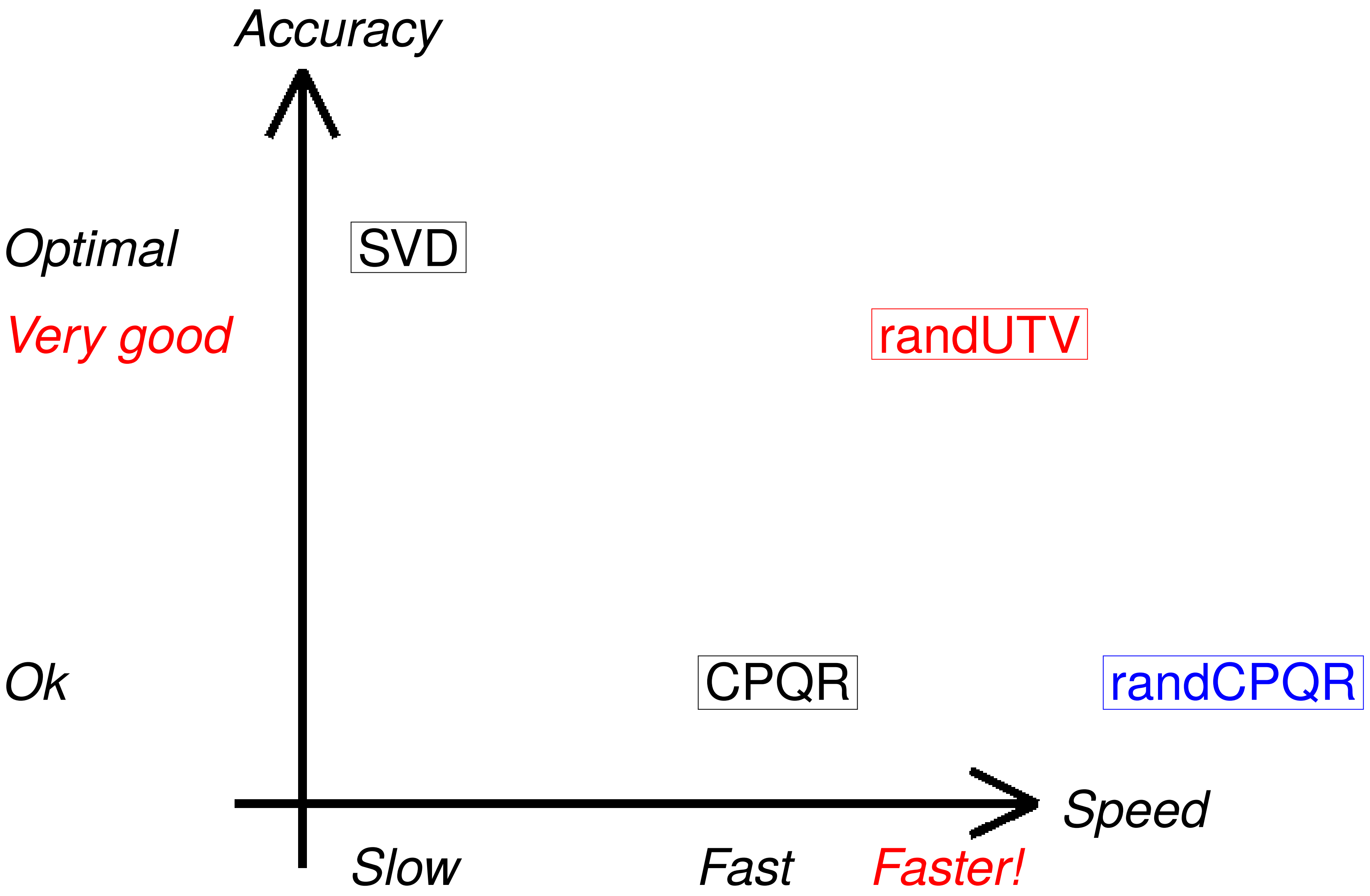
Ok CPQR

Speed

Slow *Fast*

Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: $O(n^3)$ 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 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 methods for solving $\mathbf{Ax} = \mathbf{b}$: Strassen-type methods

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

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

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

Observation:

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

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

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

Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomized 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: $O(mn^2)$

Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$: Randomized pre-conditioning

Let us consider

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

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”

$$\mathbf{X}^*\mathbf{A}\mathbf{x} = \mathbf{X}^*\mathbf{b}$$

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

$$\mathbf{X}^*\mathbf{A} = \mathbf{Q}\mathbf{R}\mathbf{P}^*.$$

Form a preconditioner

$$\mathbf{M} = \mathbf{R}\mathbf{P}^*.$$

Solve the preconditioned linear system

$$\big(\mathbf{A}\mathbf{M}^{-1}\big)\ \underbrace{\big(\mathbf{M}\mathbf{x}\big)}_{=:\mathbf{y}} = \mathbf{b}$$

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

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

Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomized 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 factorization 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 randomized solver: $O\big((\log(n) + \log(1/\varepsilon))mn + n^3\big)$.

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

Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$: Randomized 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 $\mathbf{x}_{\rm old}$.

Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomized 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)?

Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$: Randomized 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 randomized 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 $\kappa(\mathbf{A}) = \|\mathbf{A}\|_{\rm F}\,\|\mathbf{A}^{-1}\|_2$.

Randomized methods for solving $\mathbf{Ax} = \mathbf{b}$: Randomized 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 $\mathbf{X}$

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

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 . . .

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

$$\mathbf{A}^{*} = \mathbf{A}^{*}\mathbf{A}\mathbf{B}. \tag{3}$$

Equation (3) 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.

Randomized iterative solvers is a *very* active area: J. Weare & L.-H. Lim article in SIAM Review (2017); recent and current work by H. Avron, P. Drineas, M. Mahoney, D. Needell, V. Rokhlin, S. Toledo, J. Tropp, R. Ward, and many more.

Randomized methods for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$: Graph Laplacians

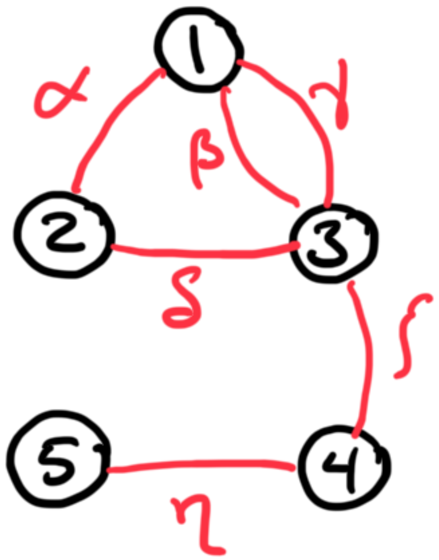
Let us consider a linear system

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

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).

Randomized 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 factorization

  $$\mathbf{A} \approx \mathbf{CC}^*,$$

  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.

Randomized 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.

Randomized 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.*

Randomized approximation of “rank-structured” matrices

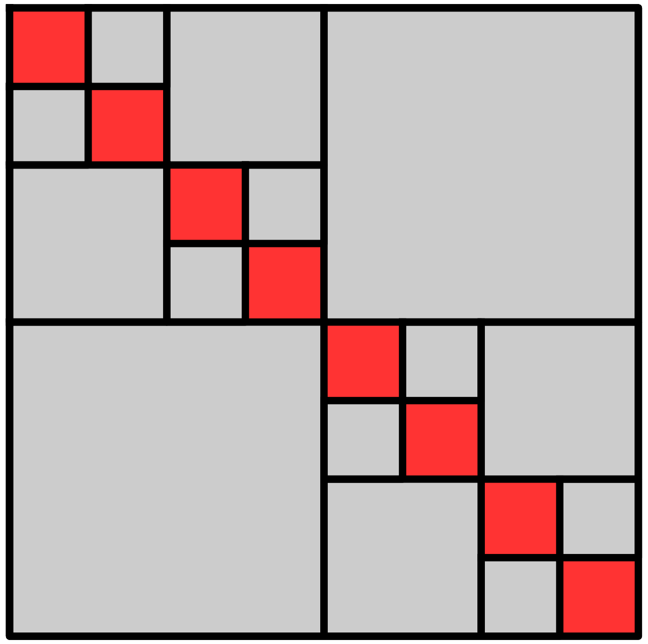
Many matrices in applications have *off-diagonal blocks* that are of low rank:

- 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 randomized 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.



Existing randomized compression schemes for rank-structured matrices:

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

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 $\mathbf{A}^{*}$.

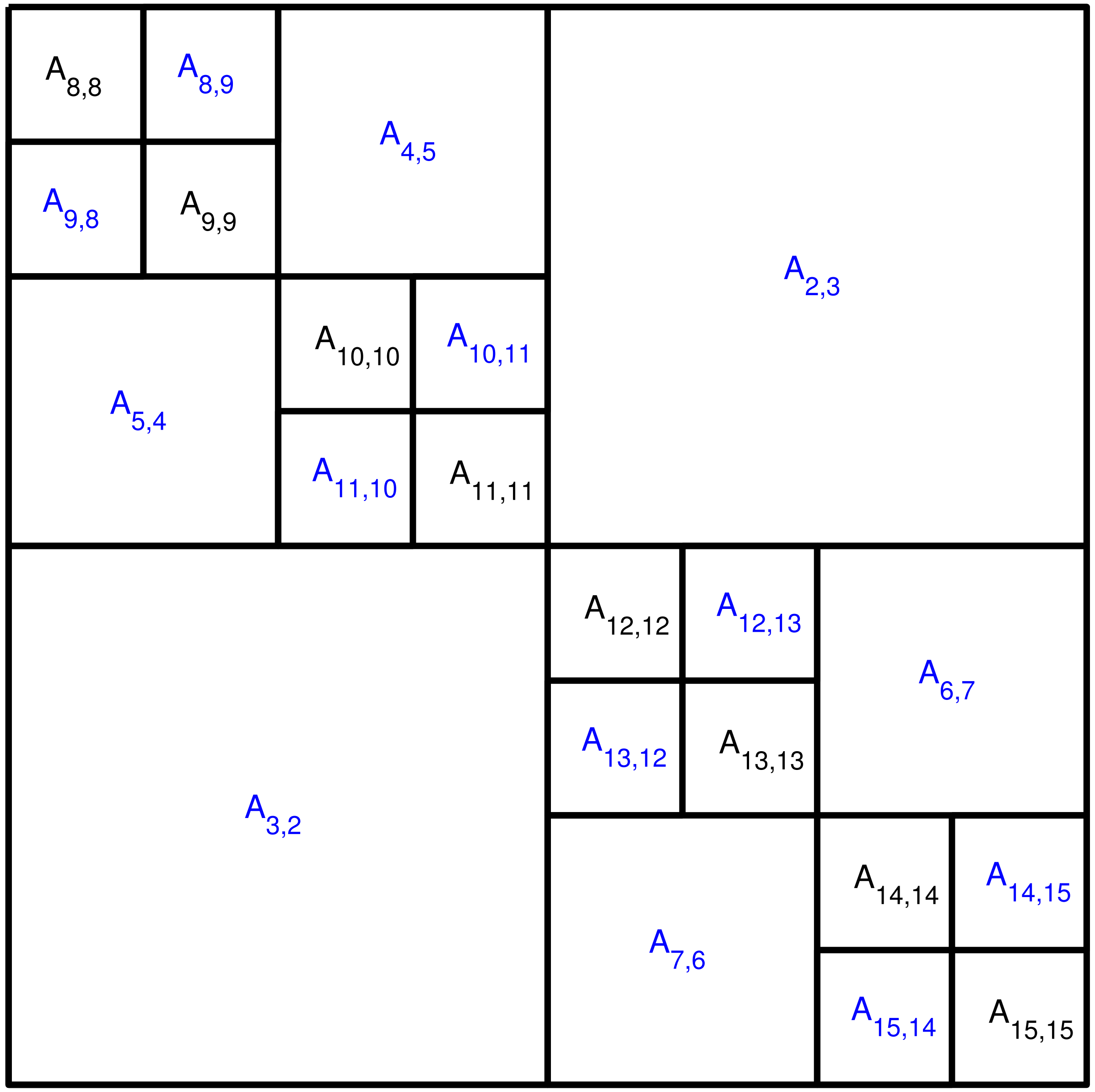
The method is still fast in many situations, and does save messy coding work. For instance, without this black-box method, implementing the matrix-matrix multiplication, or changing the partition tree, are quite hard to implement efficiently.

- 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.

“Hierarchically Off-Diagonal Low Rank (HODLR)” matrices

This is a *very simple* structured matrix format.

Informally, it means that you tessellate a matrix “like this”:



Then **A** is HODLR if each off-diagonal block (in blue in the figure) has “low rank”.

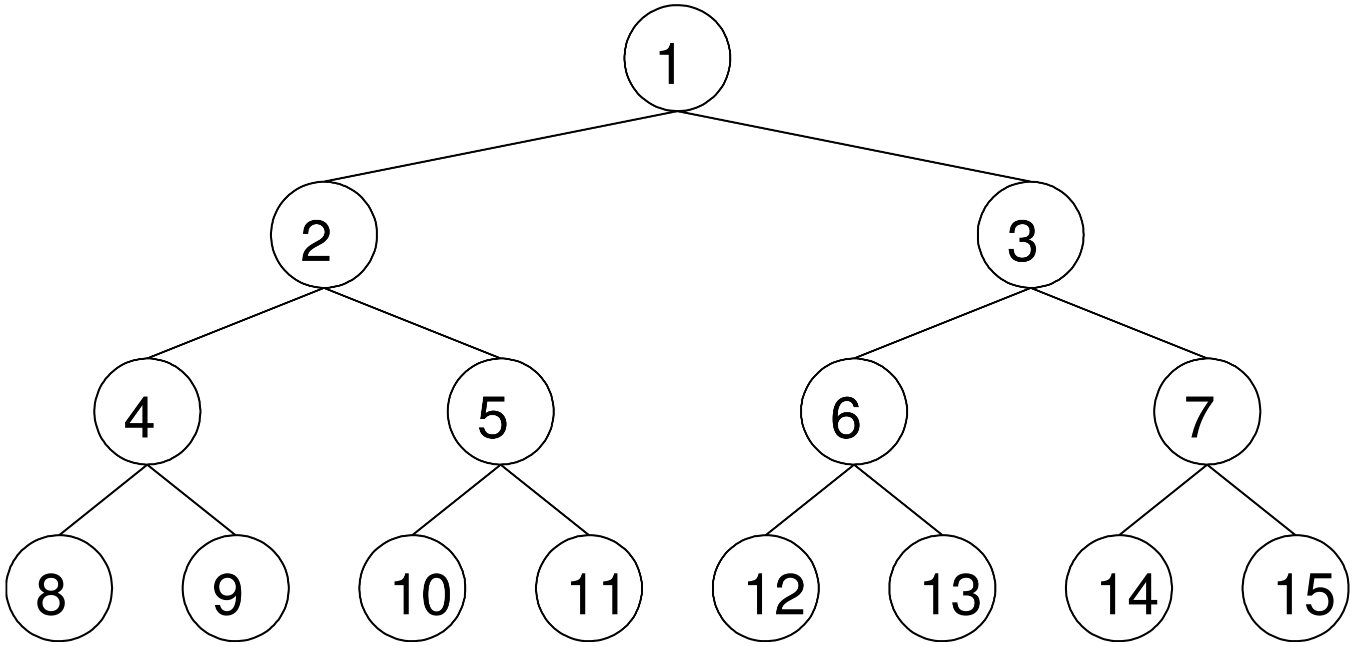
Note: A.k.a. “S-matrix” (“S” for simple), $\mathcal{H}$-matrix with weak permissibility condition, etc.

To more formally define the HODLR format, we need to introduce a *tree structure* on the index set $I = [1, 2, 3, \ldots, N]$.

Let $\mathbf{A}$ be an $N \times N$ matrix.

Suppose $\mathcal{T}$ is a binary tree on the index vector $I = [1, 2, 3, \ldots, N]$.

For a node τ in the tree, let I_τ denote the corresponding index vector.



Level 0 $I_1 = [1, 2, \ldots, 400]$

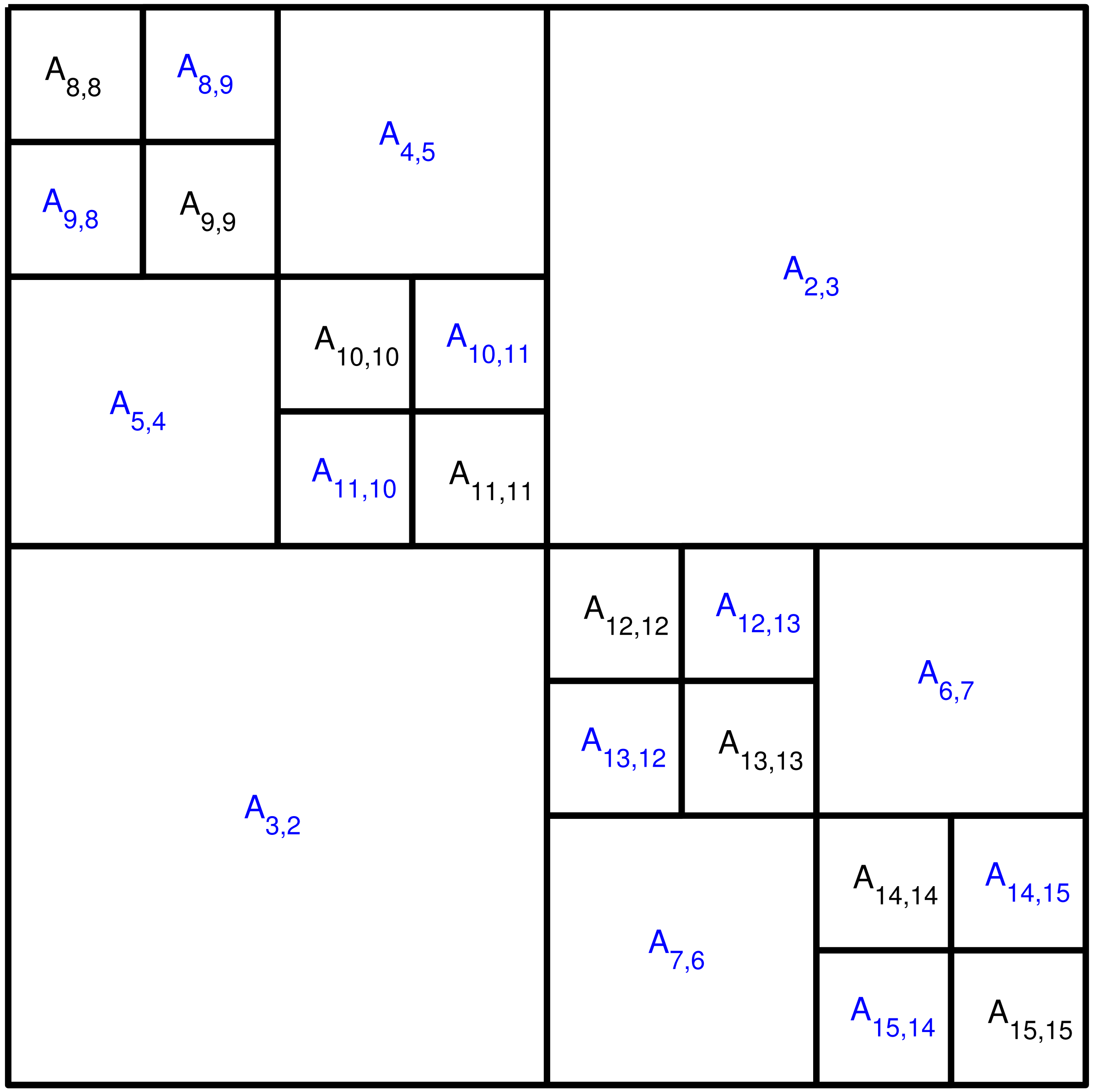
Level 1 $I_2 = [1, 2, \ldots, 200]$, $I_3 = [201, 202, \ldots, 400]$

Level 2 $I_4 = [1, 2, \ldots, 100]$, $I_5 = [101, 102, \ldots, 200]$, $\ldots$

Level 3 $I_8 = [1, 2, \ldots, 50]$, $I_9 = [51, 52, \ldots, 100]$, $\ldots$

For nodes σ and τ on the same level, set $\mathbf{A}_{\sigma,\tau} = \mathbf{A}(I_\sigma, I_\tau)$.

Formal definition of HODLR format



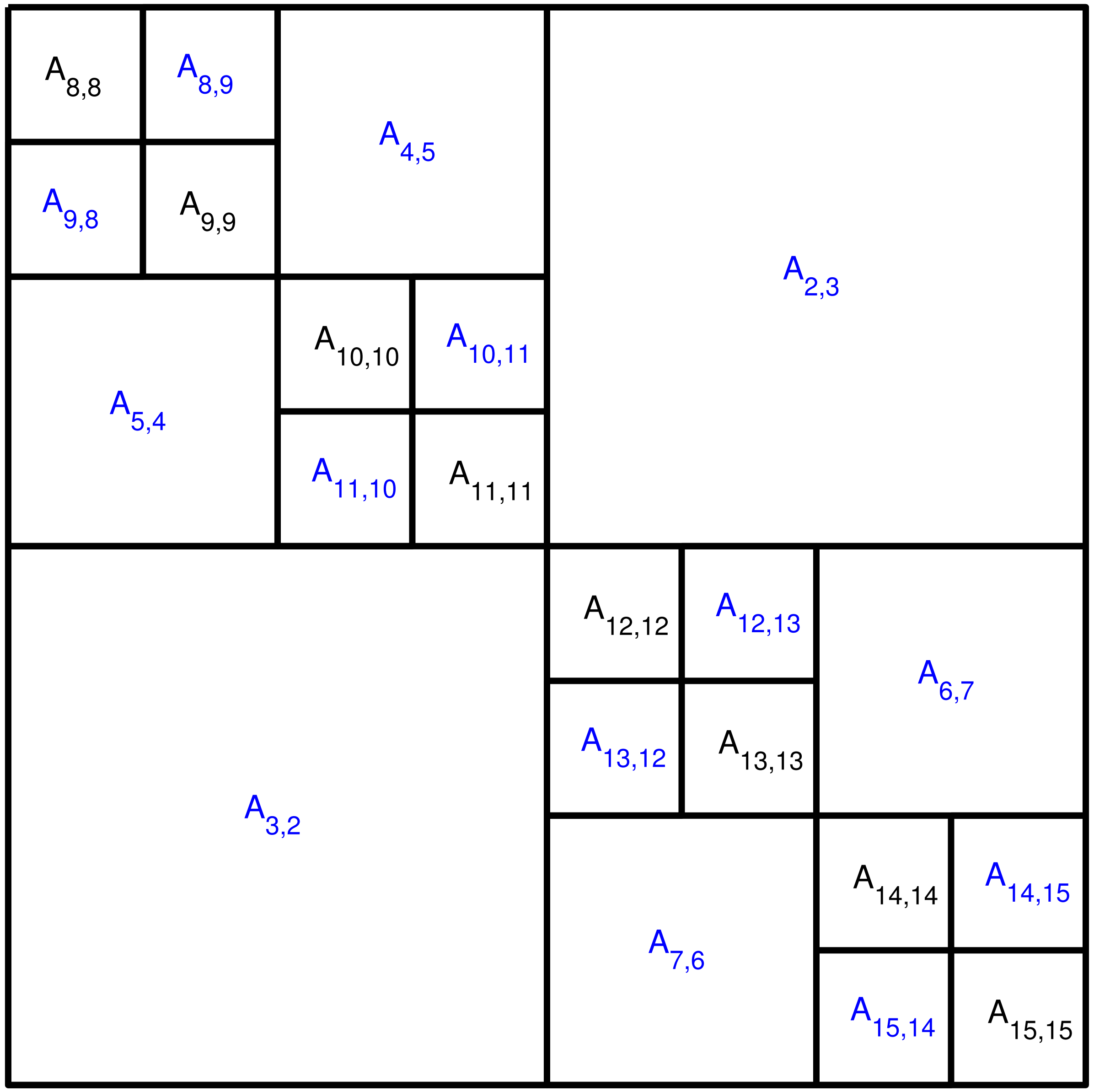
For every sibling pair $\{\alpha, \beta\}$ set

$$\mathbf{A}_{\alpha,\beta} = \mathbf{A}(I_\alpha, I_\beta).$$

Fix a bound k on the rank, and a tolerance ε. We then require that each off-diagonal block (in blue in the figure) have ε-rank at most k. Define factors

$$\begin{array}{ccccc} \mathbf{A}_{\alpha,\beta} & = & \mathcal{U}_\alpha & \tilde{\mathbf{A}}_{\alpha,\beta} & \mathcal{V}^*_\beta. \\ n_\alpha \times n_\beta & & n_\alpha \times k & k \times k & k \times n_\beta \end{array}$$

Formal definition of HODLR format



For every sibling pair $\{\alpha, \beta\}$ set

$$\mathbf{A}_{\alpha,\beta} = \mathbf{A}(I_\alpha, I_\beta).$$

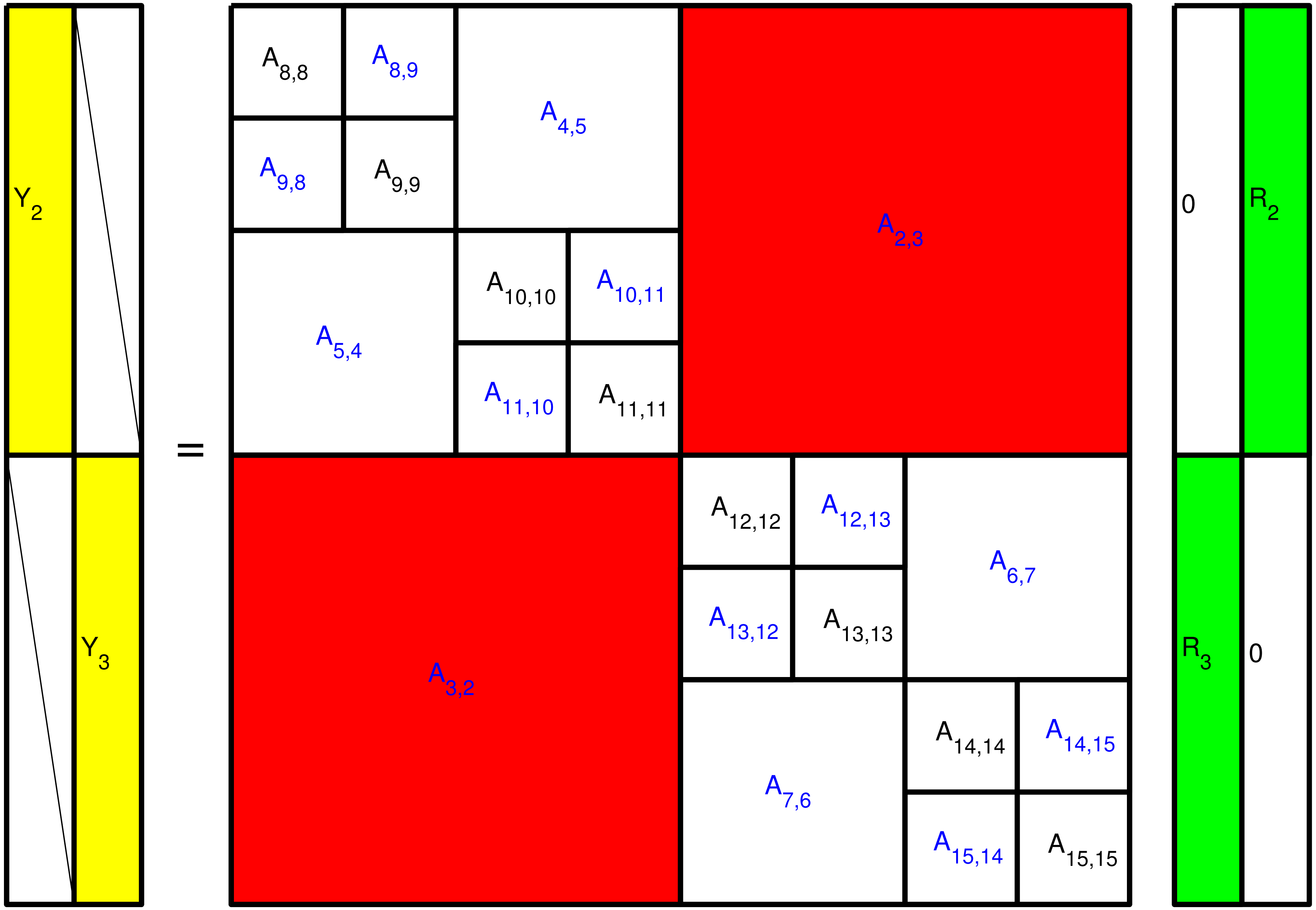
Fix a bound k on the rank, and a tolerance ε. We then require that each off-diagonal block (in blue in the figure) have ε-rank at most k. Define factors

$$\begin{array}{cccc} \mathbf{A}_{\alpha,\beta} & = \quad \mathcal{U}_\alpha & \tilde{\mathbf{A}}_{\alpha,\beta} & \mathcal{V}_\beta^*. \\ n_\alpha \times n_\beta & n_\alpha \times k & k \times k & k \times n_\beta \end{array}$$

The cost of performing a matvec is then

$$\underbrace{2 \times \frac{N}{2}k + 4 \times \frac{N}{4}k + 8 \times \frac{N}{8}k + \cdots}_{\log N \text{ terms}} \sim N \log(N)\, k.$$

The “Peeling algorithm” — level 0



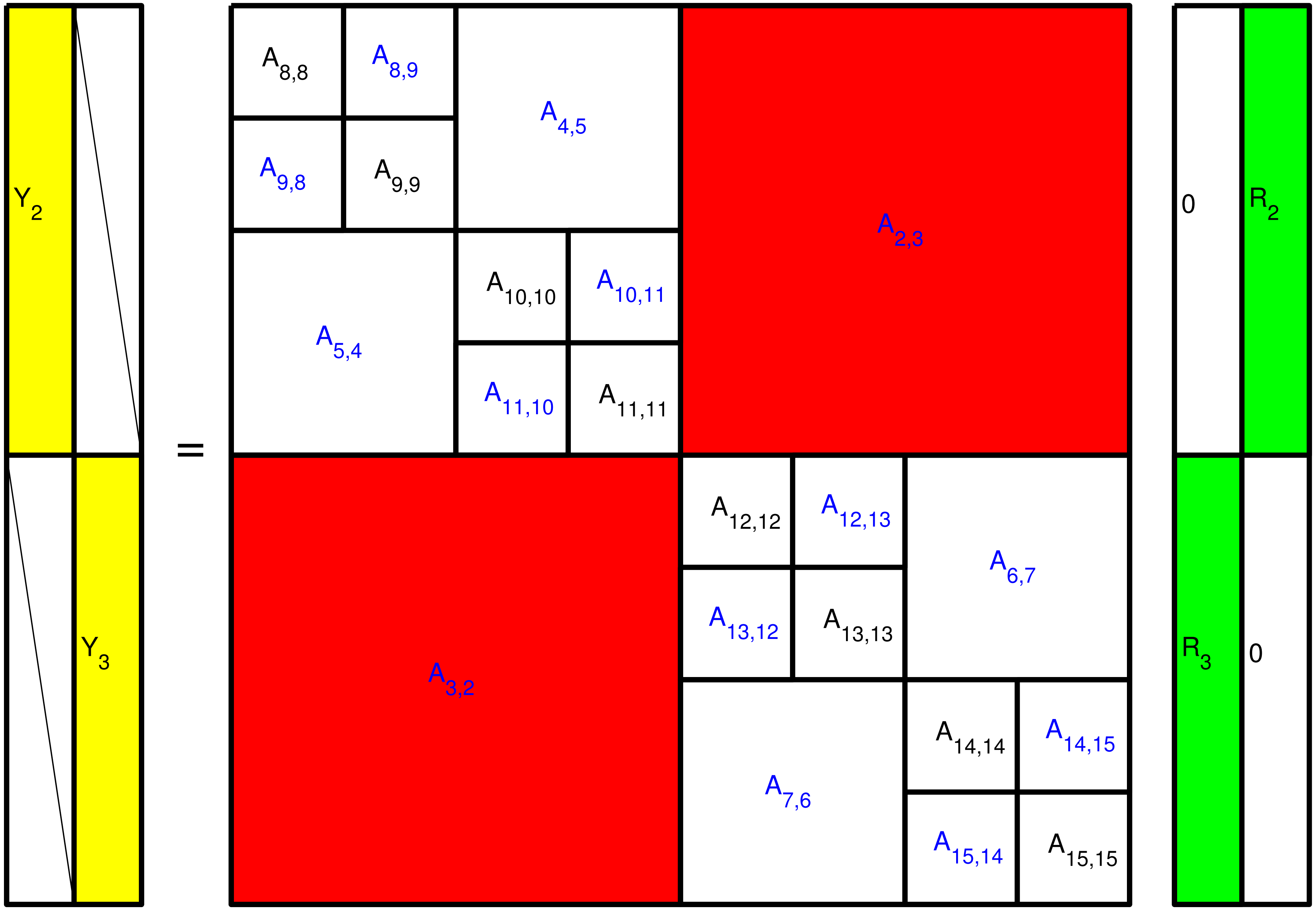
At level 0, we seek to factorize the red blocks.

Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

Now: $\mathbf{Y}_2 = \mathbf{A}_{2,3}\mathbf{R}_3$ and $\mathbf{Y}_3 = \mathbf{A}_{3,2}\mathbf{R}_2$.

The columns of $\mathbf{Y}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\mathcal{U}_\tau = \mathtt{qr}(\mathbf{Y}_\tau)$.

The “Peeling algorithm” — level 0



At level 0, we seek to factorize the red blocks.

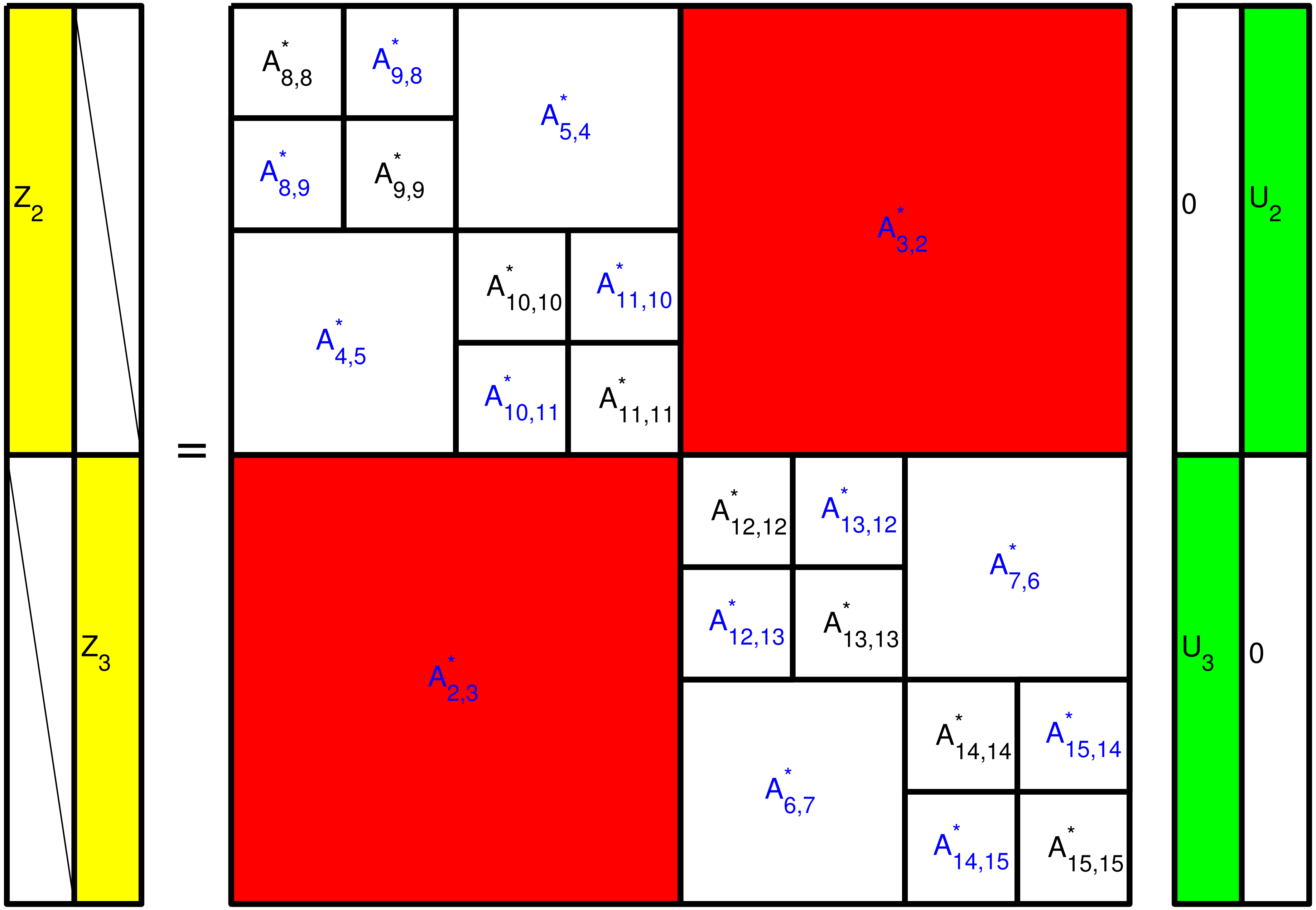
Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

Now: $\mathbf{Y}_2 = \mathbf{A}_{2,3}\mathbf{R}_3$ and $\mathbf{Y}_3 = \mathbf{A}_{3,2}\mathbf{R}_2$.

The columns of $\mathbf{Y}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\boldsymbol{\mathcal{U}}_\tau = \texttt{qr}(\mathbf{Y}_\tau)$.

How determine $\tilde{\mathbf{A}}_{2,3}$, $\tilde{\mathbf{A}}_{3,2}$, $\boldsymbol{\mathcal{V}}_2$ and $\boldsymbol{\mathcal{V}}_3$?

The “Peeling algorithm” — level 0



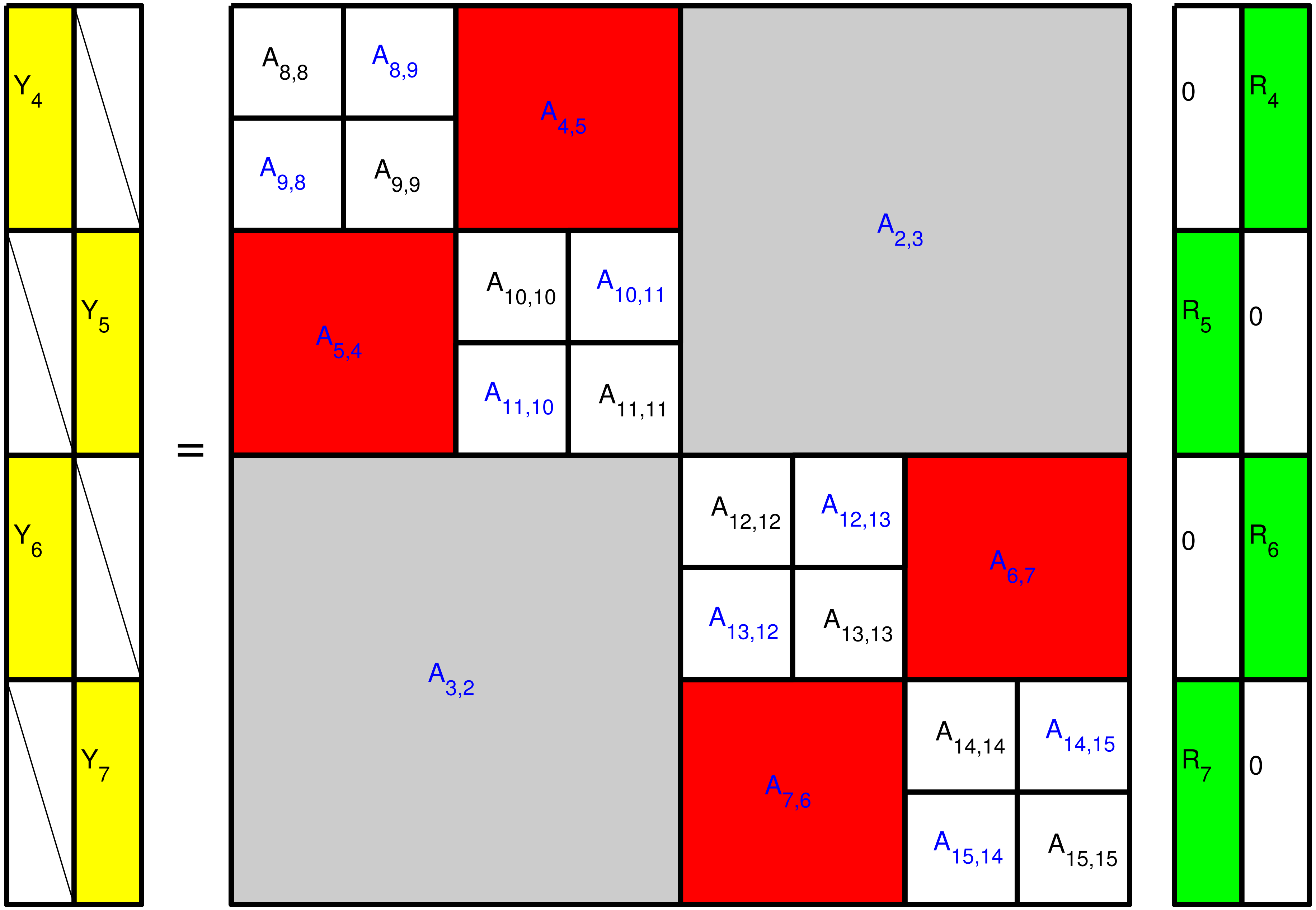
We know $\mathcal{U}_2$ and $\mathcal{U}_3$, and seek to determine $\tilde{\mathbf{A}}_{2,3}$, $\tilde{\mathbf{A}}_{3,2}$, $\mathcal{V}_2$ and $\mathcal{V}_3$.

Put $\mathcal{U}_2$ and $\mathcal{U}_3$ into the probing matrix.

Then: $\mathbf{Z}_2 = \mathbf{A}^*_{3,2}\mathcal{U}_3 = \mathcal{V}_2\mathbf{A}^*_{3,2}$ and $\mathbf{Z}_3 = \mathbf{A}^*_{2,3}\mathcal{U}_2 = \mathcal{V}_3\mathbf{A}^*_{2,3}$.

We then get $[\mathcal{V}_2, \tilde{\mathbf{A}}^*_{3,2}] = \texttt{qr}(\mathbf{Z}_2)$ and $[\mathcal{V}_3, \tilde{\mathbf{A}}^*_{2,3}] = \texttt{qr}(\mathbf{Z}_3)$.

The “Peeling algorithm” — level 1



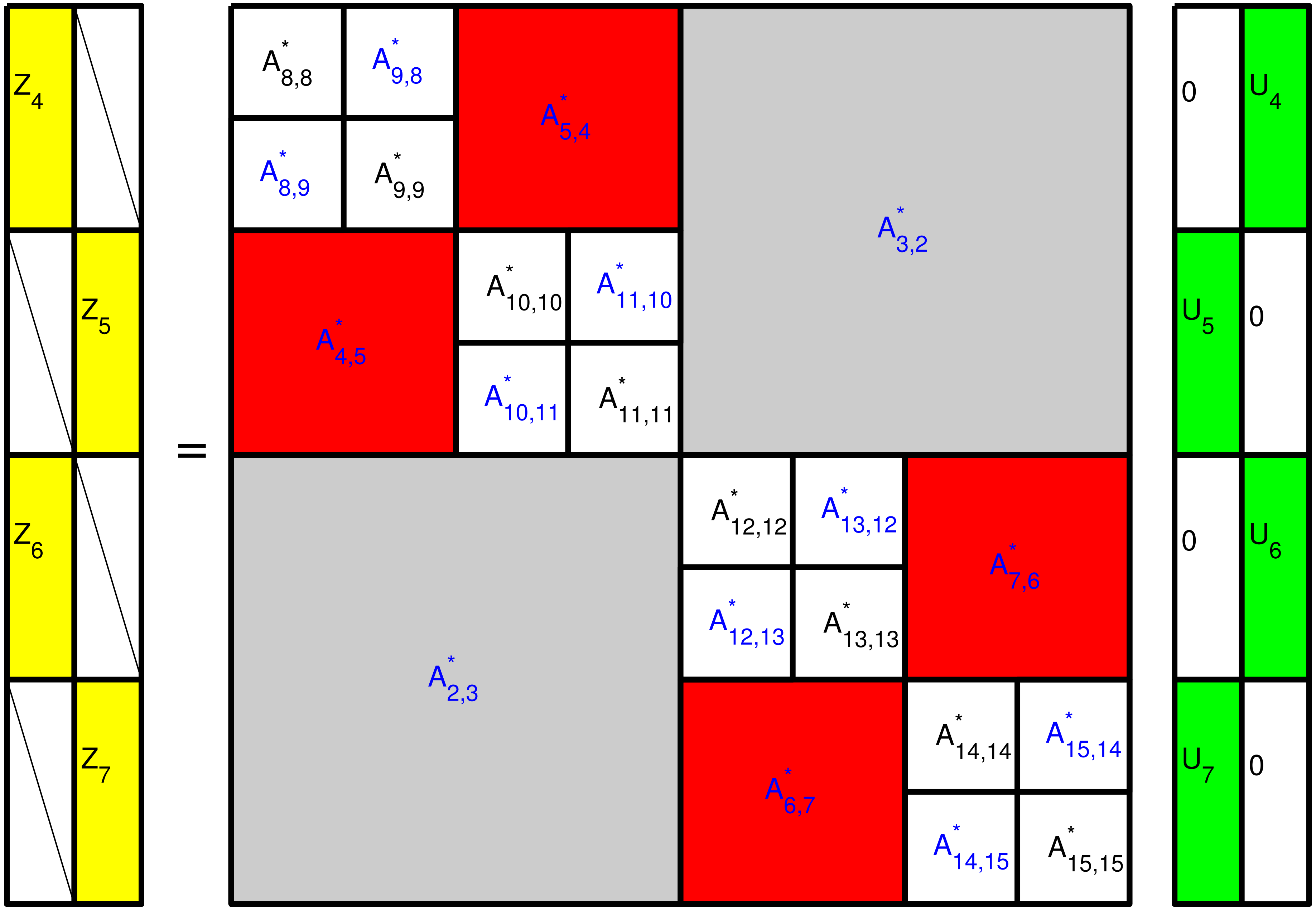
At level 1, we seek to determine the red blocks.

Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

*Subtract the contributions from the **known** gray blocks to get new sample matrices $\mathbf{Z}_\tau$.*

The columns of $\mathbf{Z}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\mathcal{U}_\tau = \texttt{orth}(\mathbf{Y}_\tau, \varepsilon)$.

The “Peeling algorithm” — level 1

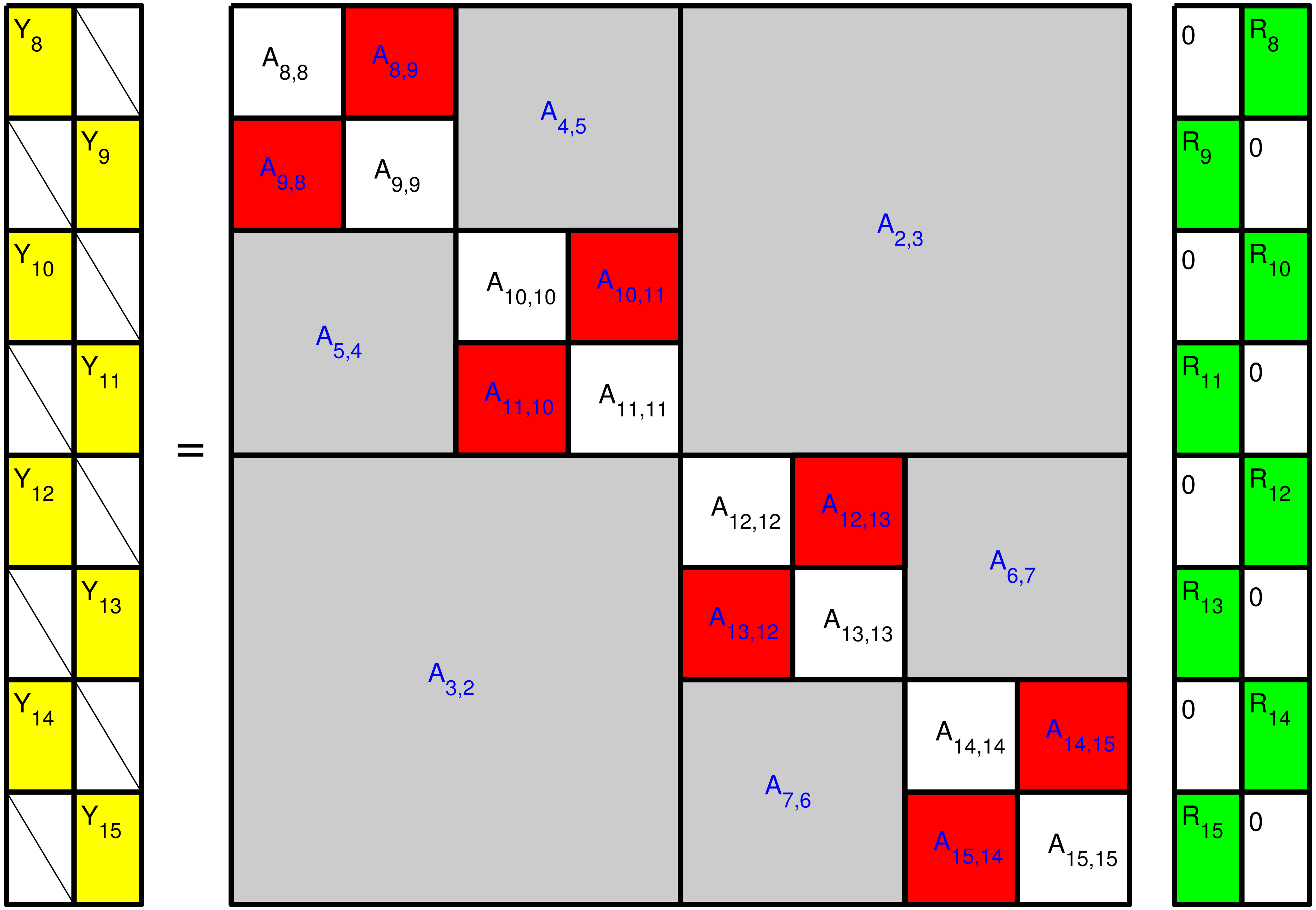


Put the matrices $\{\mathcal{U}_\tau\}_\tau$ is on level 1 into the probing matrix.

Then, e.g., $\mathbf{Z}_4 = \mathbf{A}^*_{5,4}\mathcal{U}_5 = \mathcal{V}_4\mathbf{A}^*_{5,4}$.

We get, e.g., $[\mathcal{V}_4, \tilde{\mathbf{A}}^*_{5,4}] = \mathtt{qr}(\mathbf{Z}_4, 0)$.

The “Peeling algorithm” — level 2



At level 2, we seek to determine the red blocks.

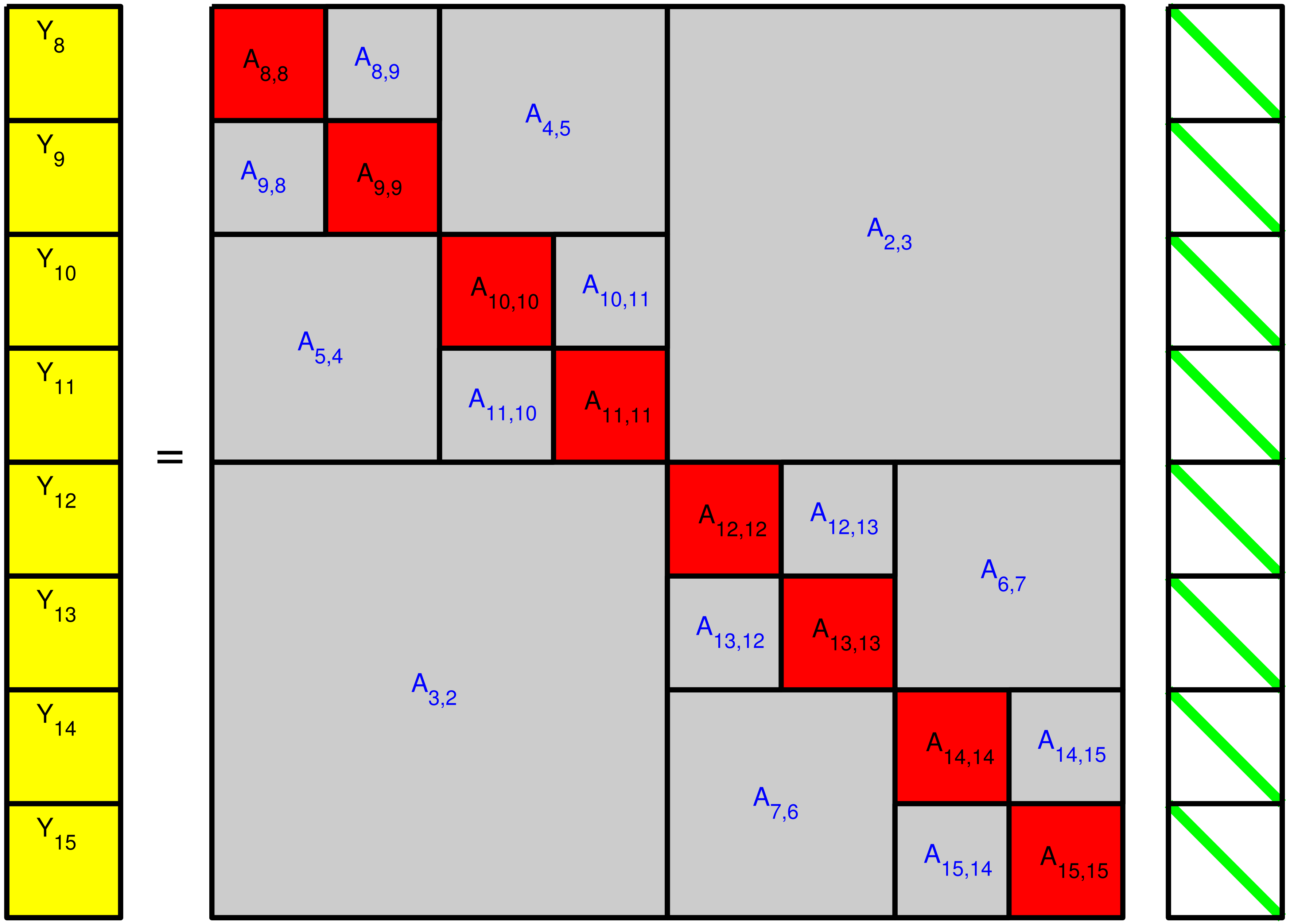
Green blocks are random matrices, yellow blocks are sample matrices $\mathbf{Y}_\tau$.

Subtract the contributions from the ***known*** *gray blocks to get new sample matrices* $\mathbf{Z}_\tau$*.*

The columns of $\mathbf{Y}_\tau$ span $\mathbf{A}(I_\tau, I_\sigma)$, where σ is the sibling of τ, so $\mathbf{U}_\tau = \texttt{orth}(\mathbf{Y}_\tau, \varepsilon)$.

By sampling $\mathbf{A}^*$, we determine $\mathbf{V}_\tau$ and $\tilde{\mathbf{A}}_{\tau,\sigma}$.

The “Peeling algorithm” — leaves



Once you reach the finest level, simply use small identity matrices as probing matrices.

The diagonal blocks $\mathbf{D}_\tau$ are obtained by subtracting contributions from the off-diagonal blocks from $\mathbf{Y}_\tau$.

```
Build compressed representations of all off-diagonal blocks.
loop over levels $\ell = 0 : (L-1)$
    Build the random matrices $\mathbf{R}_1$ and $\mathbf{R}_2$.
    $\mathbf{R}_1 = \mathtt{zeros}(n,r)$
    $\mathbf{R}_2 = \mathtt{zeros}(n,r)$
    loop over boxes $\tau$ on level $\ell$
        Let $\{\alpha,\beta\}$ denote the children of box $\tau$.
        $\mathbf{R}_1(I_\alpha,:) = \mathtt{randn}(n_\alpha,r)$
        $\mathbf{R}_2(I_\beta,:) = \mathtt{randn}(n_\beta,r)$
    end loop
    Apply $\mathbf{A}$ to build the samples for the
    incoming basis matrices.
    $\mathbf{Y}_1 = \mathbf{A}\mathbf{R}_2 - \mathbf{A}^{(\ell)}\mathbf{R}_2$
    $\mathbf{Y}_2 = \mathbf{A}\mathbf{R}_1 - \mathbf{A}^{(\ell)}\mathbf{R}_1$
    Orthonormalize the sample matrices to build
    the incoming basis matrices.
    loop over boxes $\tau$ on level $\ell$
        Let $\{\alpha,\beta\}$ denote the children of box $\tau$.
        $\mathcal{U}_\alpha = \mathtt{qr}(\mathbf{Y}_1(I_\alpha,:))$.
        $\mathcal{U}_\beta = \mathtt{qr}(\mathbf{Y}_2(I_\beta,:))$.
        $\mathbf{R}_1(I_\alpha,:) = \mathcal{U}_\alpha$
        $\mathbf{R}_2(I_\beta,:) = \mathcal{U}_\beta$
    end loop
    Apply $\mathbf{A}^*$ to build the samples for the outgoing basis matrices.
    $\mathbf{Z}_1 = \mathbf{A}^*\mathbf{R}_2 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{R}_2$
    $\mathbf{Z}_2 = \mathbf{A}^*\mathbf{R}_1 - \left(\mathbf{A}^{(\ell)}\right)^*\mathbf{R}_1$
    Take local SVDs to build incoming basis matrices and
    sibling interaction matrices.
    loop over boxes $\tau$ on level $\ell$
        Let $\{\alpha,\beta\}$ denote the children of box $\tau$.
        $[\mathcal{V}_\alpha, \mathbf{B}_{\beta\alpha}, \hat{\mathbf{U}}_\beta] = \mathtt{svd}(\mathbf{Z}_1(I_\alpha,:),\varepsilon)$.
        $[\mathcal{V}_\beta, \mathbf{B}_{\alpha\beta}, \hat{\mathbf{V}}_\alpha] = \mathtt{svd}(\mathbf{Z}_2(I_\beta,:),\varepsilon)$.
        $\mathcal{U}_\beta \leftarrow \mathcal{U}_\beta\hat{\mathbf{U}}_\beta$.
        $\mathcal{U}_\alpha \leftarrow \mathcal{U}_\alpha\hat{\mathbf{U}}_\alpha$.
    end loop
end loop
Extract the diagonal matrices.
$n_{\max} = \max\{n_\tau :\ \tau \text{ is a leaf}\}$
$\mathbf{R} = \mathtt{zeros}(N, n_{\max})$
loop over leaf boxes $\tau$
    $\mathbf{R}(I_\tau, 1:n_\tau) = \mathtt{eye}(n_\tau)$.
end loop
$\mathbf{Y} = \mathbf{A}\mathbf{R} - \mathbf{A}^{(L)}\mathbf{R}$
loop over leaf boxes $\tau$
    $\mathbf{D}_\tau = \mathbf{Y}(I_\tau, 1:n_\tau)$.
end loop
```

The matrix $\mathbf{A}^{(\ell)}$ consists of all blocks on level ℓ or coarser.

Key points on “peeling” algorithm for compressing rank-structured matrices:

- Entirely black-box — very easy to use.
- Complexity $O(N \log N)$.
- “Reasonable” practical speed.

Question: Is $O(N)$ complexity (without log factors) attainable?

Key points on “peeling” algorithm for compressing rank-structured matrices:

- Entirely black-box — very easy to use.
- Complexity $O(N \log N)$.
- “Reasonable” practical speed.

Question: Is $O(N)$ complexity (without log factors) attainable?

Answer: Yes, but currently with many caveats. Most importantly, current methods require the ability to evaluate a small number of individual entries of the matrix.

Key points on “peeling” algorithm for compressing rank-structured matrices:

- Entirely black-box — very easy to use.
- Complexity $O(N \log N)$.
- “Reasonable” practical speed.

Question: Is $O(N)$ complexity (without log factors) attainable?

Answer: Yes, but currently with many caveats. Most importantly, current methods require the ability to evaluate a small number of individual entries of the matrix.

Research objectives:

- Accelerate randomized compression algorithms for rank-structured matrices.
- Attempt to build $O(N)$ methods that do *not* require individual entries.
  This appears to be non-trivial ...
- Improve performance for non-uniform trees.
- Improve accuracy by incorporating power iteration, if needed.
- Eliminate tuning parameters.

Key points:

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

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

- Recommended approach: *Sketch first, then return to original data.*

  (As opposed to: Sketch, then solve using only the sketched data.)

- Even though the algorithms are randomized, *the output can be trusted.*

  The probability of failure is *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 factorizations of matrices.*

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

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

2. *Randomized algorithms for structured matrices.*

   Use randomization 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 randomized projections to accelerate non-linear algebraic tasks.*

   Faster nearest neighbor search, faster clustering algorithms, etc. The idea is to use randomized 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 factorizations of matrices.*

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

   New “UTV” factorization method is almost as accurate as SVD and much faster.

2. *Randomized algorithms for structured matrices.*

   Use randomization 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 randomized projections to accelerate non-linear algebraic tasks.*

   Faster nearest neighbor search, faster clustering algorithms, etc. The idea is to use randomized 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.ices.utexas.edu/~pgm/main_publications.html`):

- P.G. Martinsson, J. Tropp, "Randomized algorithms for linear algebra." *Acta Numerica,* 2020.
- 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`