Randomized algorithms in linear algebra and scientific computing

Per-Gunnar Martinsson

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

University of Texas at Austin

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

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

Outline:

- **Randomized algorithms for low rank approximation**
  Randomized singular value decomposition (RSVD) and randomized embeddings.
  Fast Johnson-Lindenstrauss transforms.
  Streaming and out-of-core algorithms.

- **Other examples randomized algorithms for linear algebraic problems**
  Randomization as a pre-conditioner in linear solvers.
  Sketching to reduce data storage in scientific simulations.
  Randomized sampling methods for otherwise inaccessible problems.

- **Randomized algorithms in high performance computing**
  Las Vegas vs. Monte Carlo. How RSVD fits ("in between" but closer to LV).
  Reliability and trustworthiness of the output. Certificates of accuracy.
  Reproducibility, impact on software development, etc.

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

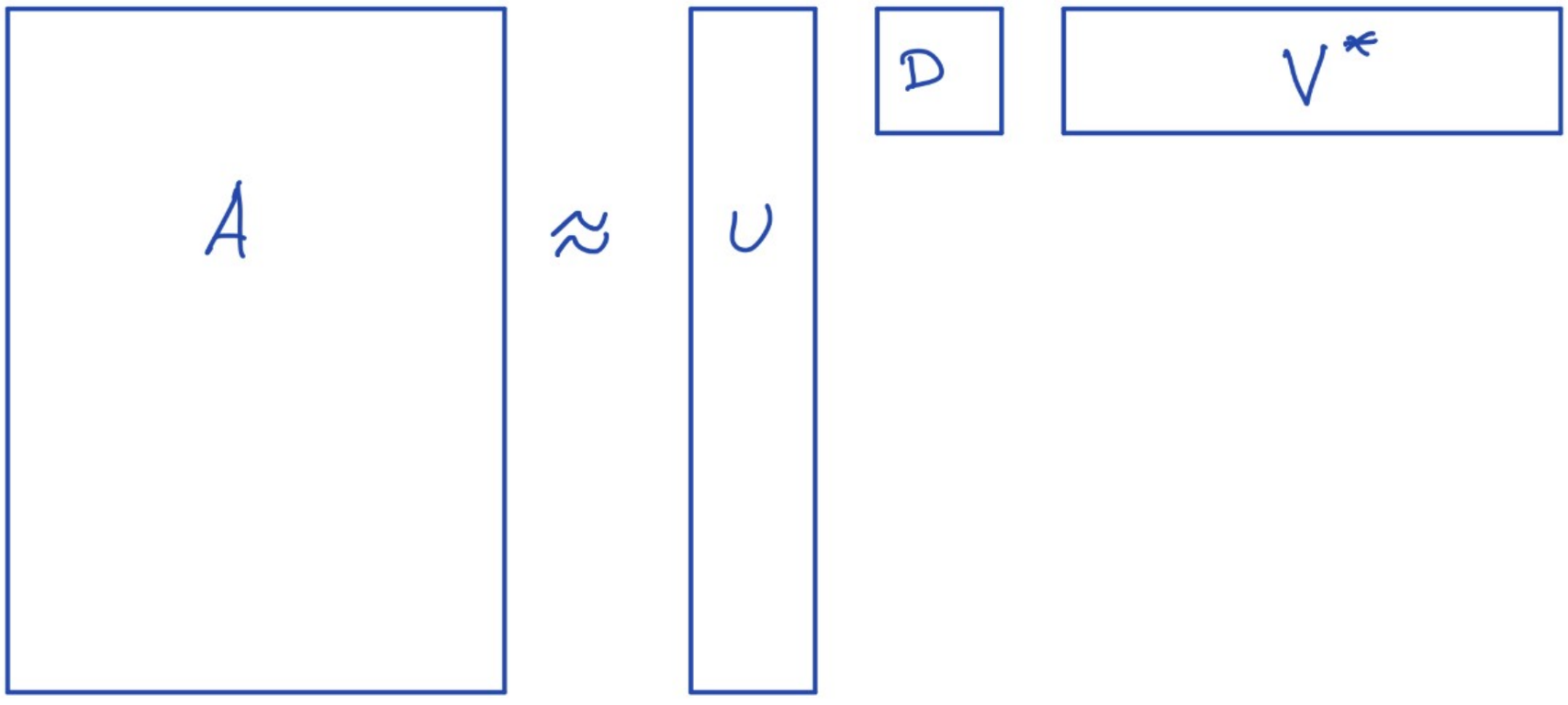
Apologies: *Citations are necessarily* ***very*** *incomplete. For more details, see surveys at* *https://arxiv.org/abs/2002.01387* **and** *https://arxiv.org/abs/0909.4061*

Randomized algorithms for low rank approximation:

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

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

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



Randomized algorithms for low rank approximation:

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

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

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

Applications:

- Principal component analysis (fitting a hyperplane to data).
- Model reduction in analyzing physical systems.
- PageRank and other spectral methods in data analysis.
- Fast algorithms in scientific computing. (FMMs, Fast Direct Solvers, etc.)
- Diffusion geometry and manifold learning.
- Many, many more ...

Classical deterministic methods: Gram-Schmidt (column pivoted QR), Krylov techniques, ...

Randomized algorithms for low rank approximation:

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

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

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

Solution:

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

Why does it work? When $\mathbf{A}$ has exact rank k, the algorithm succeeds with probability 1.

Randomized algorithms for low rank approximation:

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

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

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

Solution:

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

Why does it work? When $\mathbf{A}$ has exact rank k, the algorithm succeeds with probability 1. In the general case, it fails only if the columns of $\mathbf{G}$ manage to all be close to orthogonal to a dominant right singular vector. *Very* unlikely.

Randomized algorithms for low rank approximation:

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

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

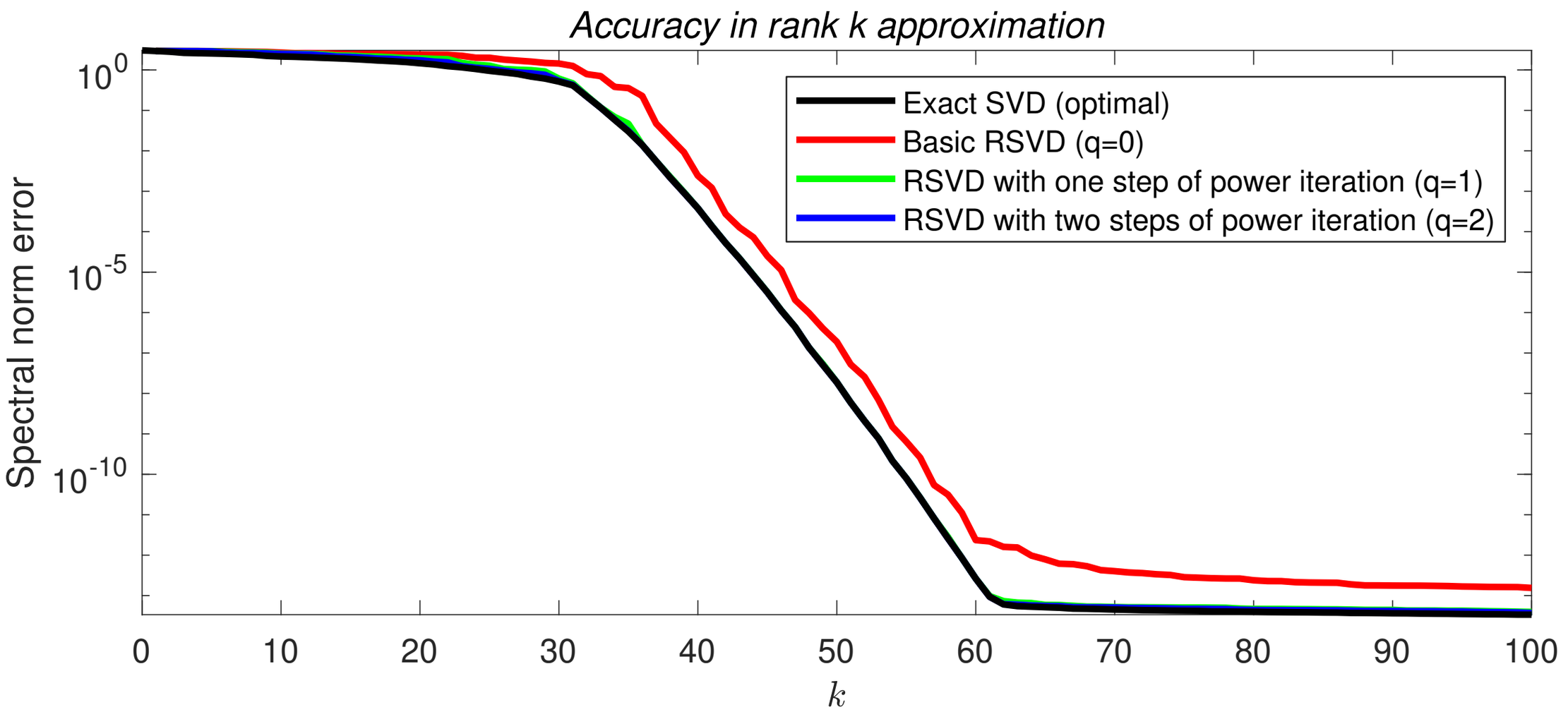
with $\mathbf{U}$ and $\mathbf{V}$ having orthonormal columns, and $\mathbf{D}$ diagonal.

Solution:

1. Draw an $n \times k$ Gaussian random matrix $\mathbf{G}$. `G = randn(n,k)`
2. Form the $m \times k$ sample matrix $\mathbf{Y} = \mathbf{AG}$. `Y = A * G`
3. Form an $m \times k$ orthonormal matrix $\mathbf{Q}$ s. t. $\mathrm{col}(\mathbf{Y}) = \mathrm{col}(\mathbf{Q})$. `[Q, ~] = qr(Y)`
4. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q' * A`
5. Compute the SVD of $\mathbf{B}$ (small!): $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat, Sigma, V] = svd(B,'econ')`
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q * Uhat`

Power iteration: When the singular values of $\mathbf{A}$ decay slowly, precision can be improved by replacing the formula $\mathbf{Y} = \mathbf{AG}$ on line 2 by $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*\mathbf{G})$, or $\mathbf{Y} = \mathbf{A}(\mathbf{A}^*(\mathbf{AG}))$, or ...

Randomized low rank approximation:



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

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

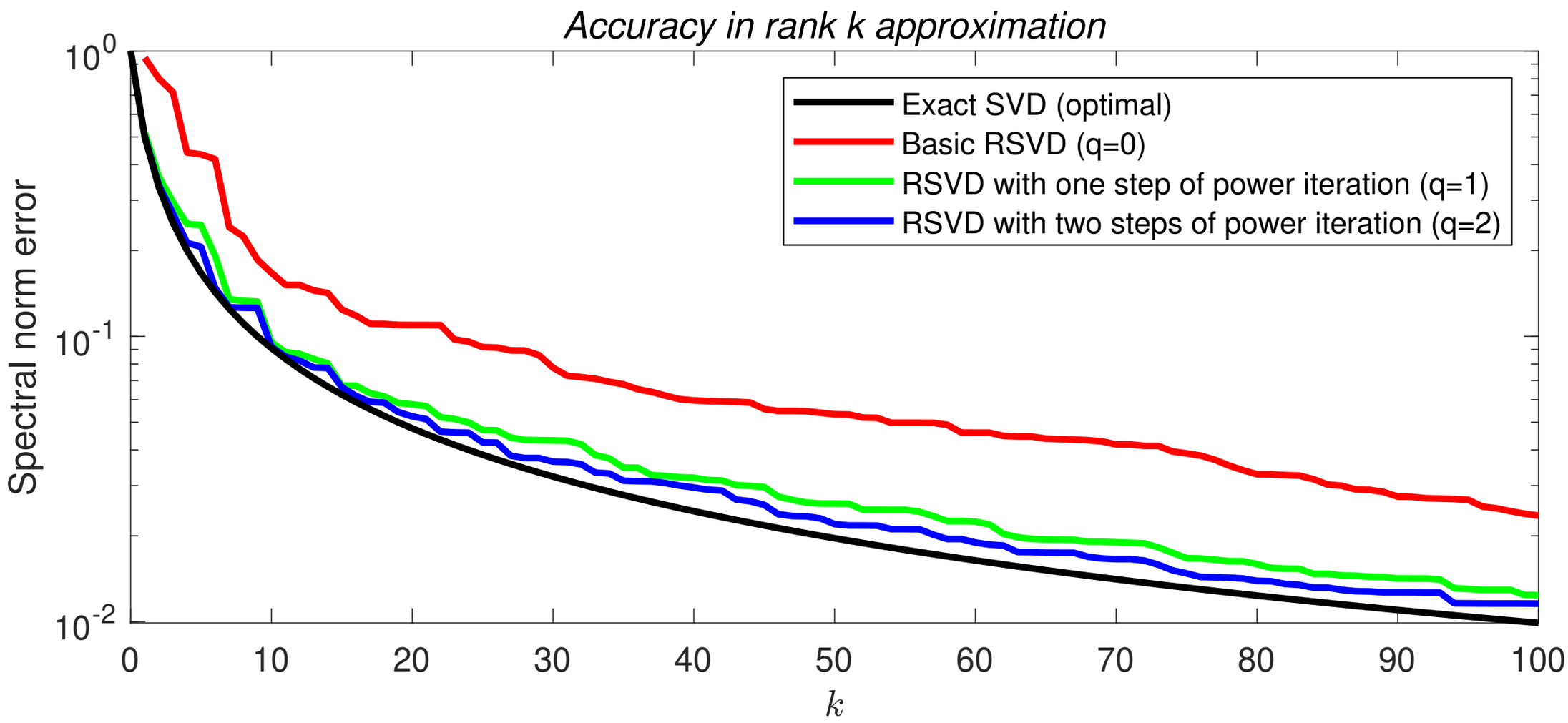
where $\mathbf{P}_k$ is the orthogonal projection onto the first k columns of

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

and where $\mathbf{G}$ is a Gaussian random matrix.

The matrix $\mathbf{A}$ is an approximation to a scattering operator for a Helmholtz problem.

Randomized low rank approximation:



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

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

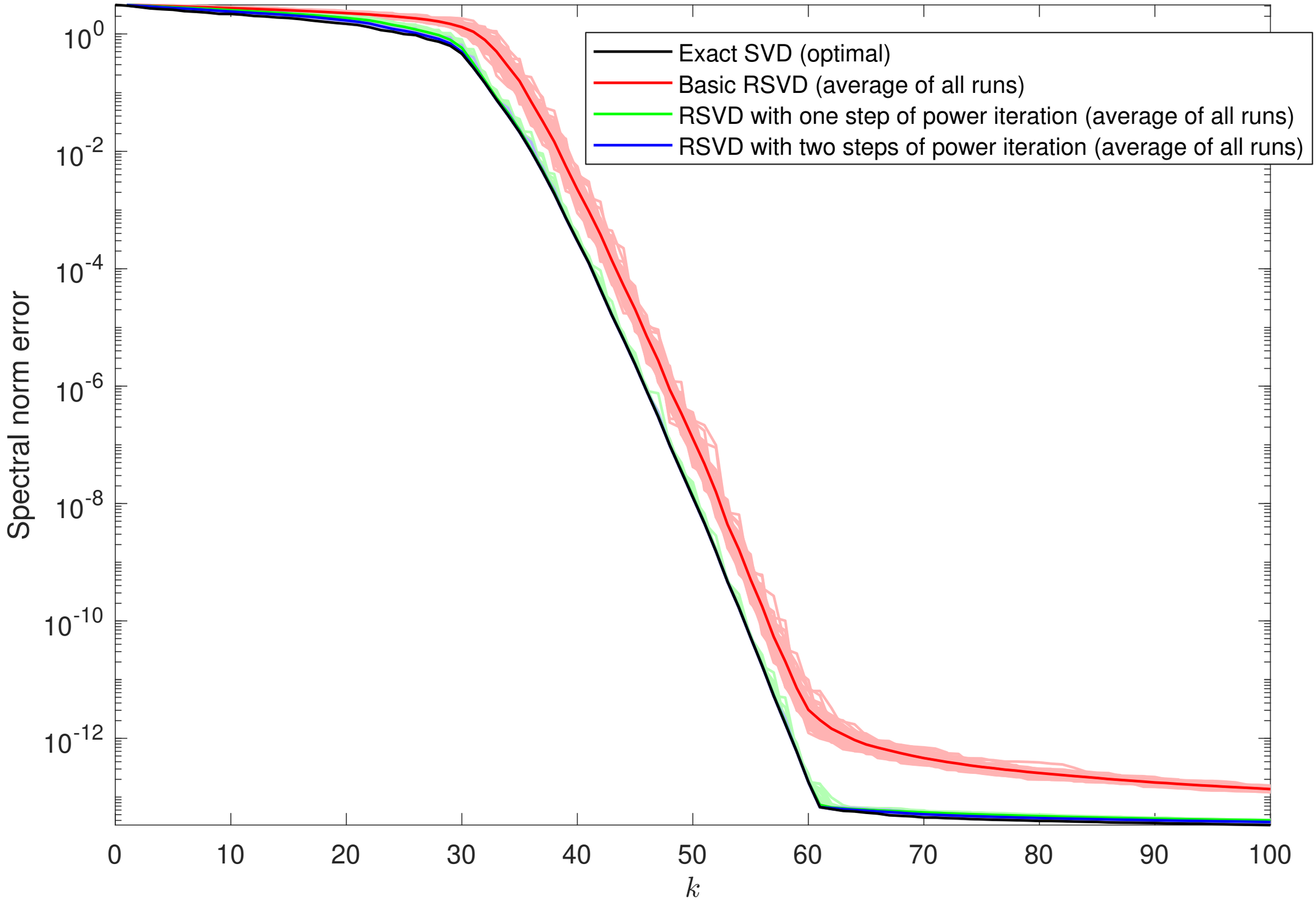
where $\mathbf{P}_k$ is the orthogonal projection onto the first k columns of

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

and where $\mathbf{G}$ is a Gaussian random matrix.

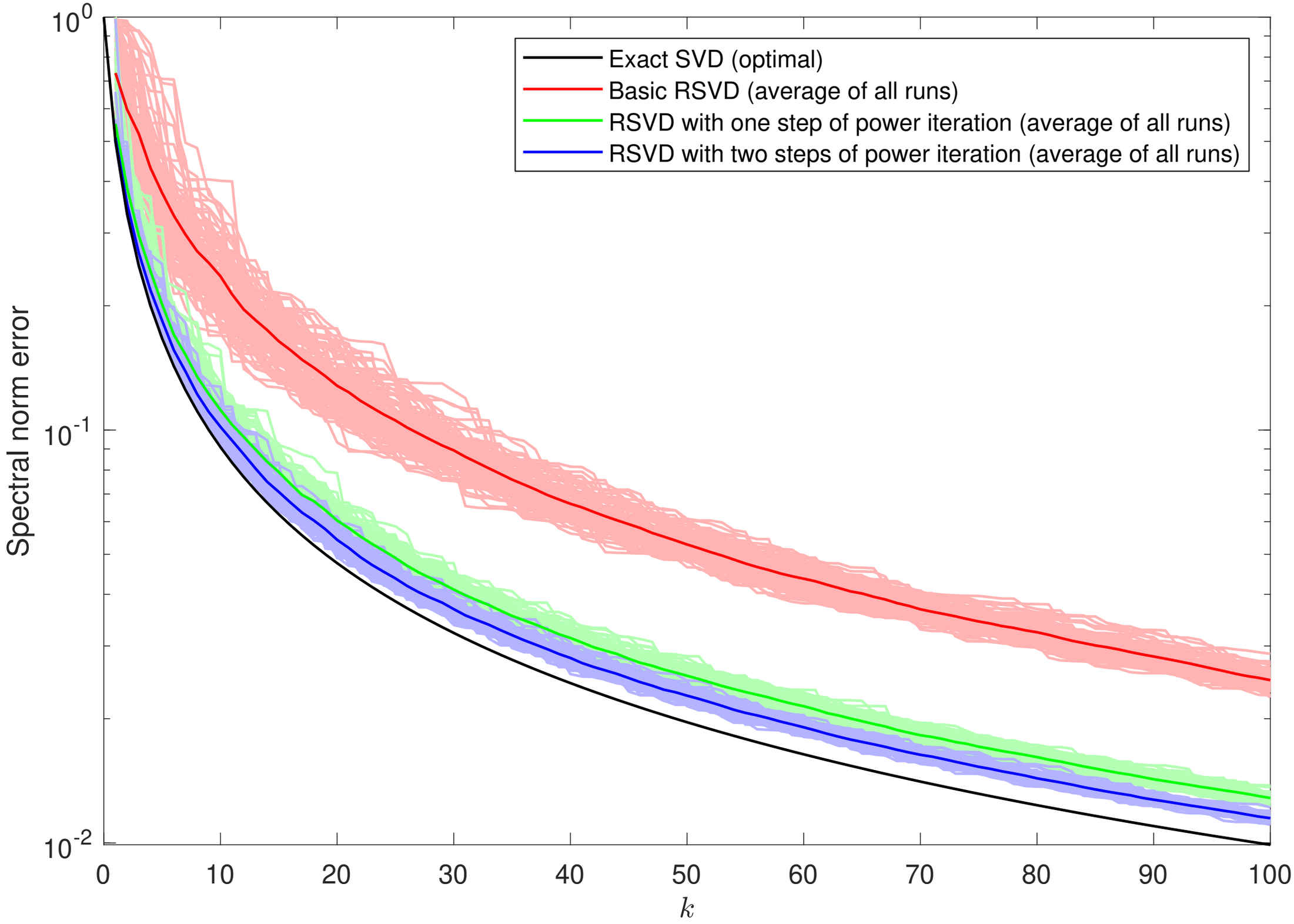
The matrix $\mathbf{A}$ now has singular values that decay slowly.

Randomized low rank approximation: The same plot, but showing 100 instantiations.



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

Randomized low rank approximation: The same plot, but showing 100 instantiations.



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

Input: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Oversampling: By drawing a small number p of extra samples, we can prove that the error is close to theoretically minimal. Think $p = 5$ or $p = 10$.

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

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

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

Oversampling: By drawing a small number p of extra samples, we can prove that the error is close to theoretically minimal. Think $p = 5$ or $p = 10$. For instance, we have:

Theorem: [Halko, Martinsson, Tropp, 2009 & 2011] Let $\mathbf{A}$ denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Let k denote a target rank and let p denote an over-sampling parameter. Let $\mathbf{G}$ denote an $n \times (k+p)$ Gaussian matrix. Let $\mathbf{Q}$ denote the $m \times (k+p)$ matrix $\mathbf{Q} = \texttt{orth}(\mathbf{AG})$. If $p \geq 2$, then

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

and

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

There are also bounds on the error when power iteration is used, the likelihood of large deviations from the expected value, and so on.

Input: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key results on randomized SVD:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.

Input: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix $\mathbf{G}$.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix $\mathbf{Q}$ s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key results on randomized SVD:

- High practical speed — interacts with $\mathbf{A}$ only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).

Input: An $m \times n$ matrix **A**, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key results on randomized SVD:

- High practical speed — interacts with **A** only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn\log k)$.

  The key is to use a *Fast Johnson-Lindenstrauss transform*.

  Practical acceleration is achieved at ordinary matrix sizes.

Input: An $m \times n$ matrix **A**, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key results on randomized SVD:

- High practical speed — interacts with **A** only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn \log k)$.
- Single pass algorithms have been developed for *streaming environments.*

  The idea is that you are allowed to observe each matrix element only once.

  You cannot store the matrix.

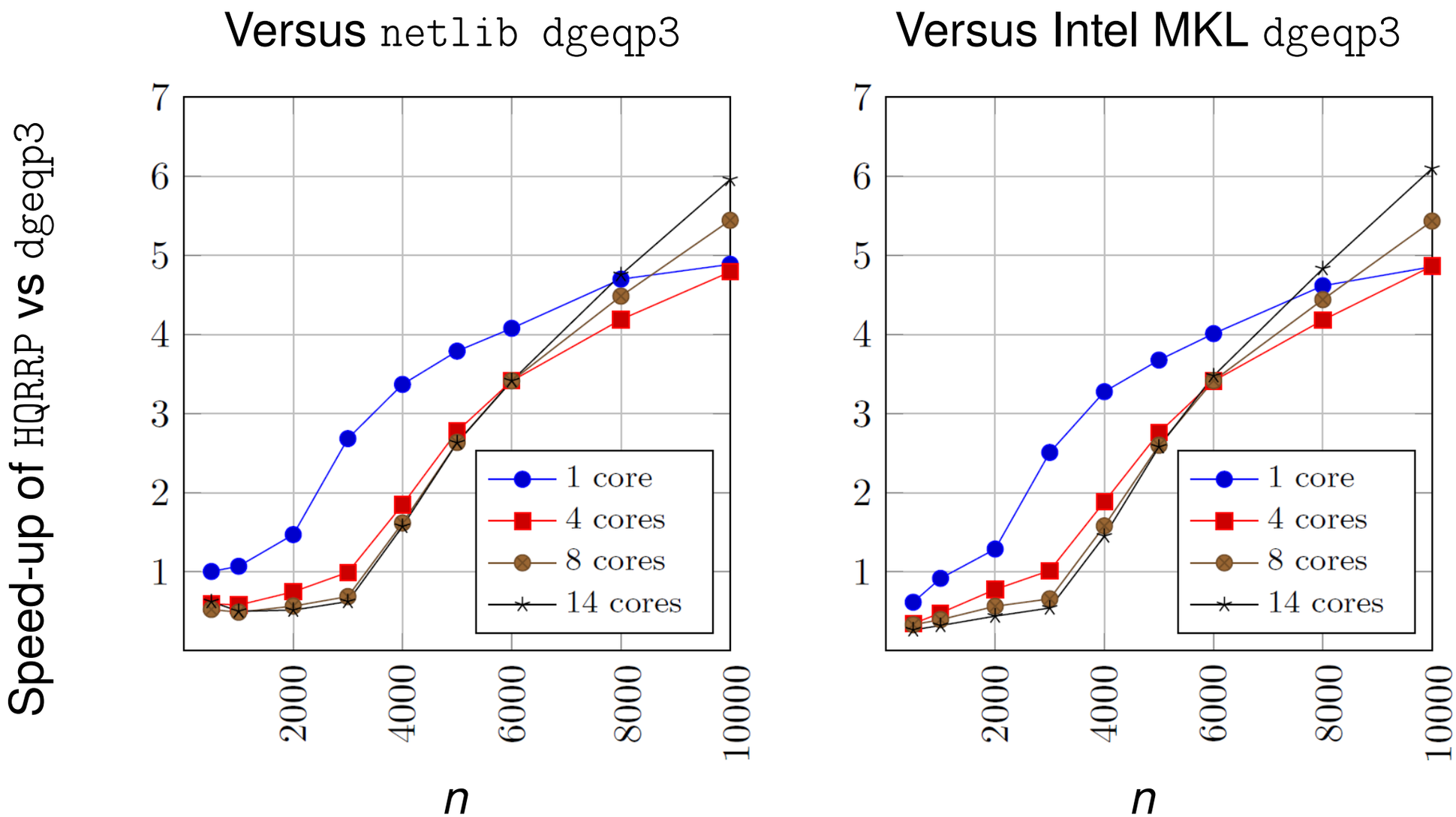
  Not possible with deterministic methods!

Input: An $m \times n$ matrix **A**, a target rank k, and an over-sampling parameter p (say $p = 5$).	
Output: Rank-$(k+p)$ factors **U**, **D**, and **V** in an approximate SVD $\mathbf{A} \approx \mathbf{UDV}^*$.	
(1) Draw an $n \times (k+p)$ random matrix **G**.	(4) Form the small matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$.
(2) Form the $m \times (k+p)$ sample matrix $\mathbf{Y} = \mathbf{AG}$.	(5) Factor the small matrix $\mathbf{B} = \hat{\mathbf{U}}\mathbf{DV}^*$.
(3) Compute an ON matrix **Q** s.t. $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$.	(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

Key results on randomized SVD:

- High practical speed — interacts with **A** only through matrix-matrix multiplication.
- Order of magnitude acceleration for data stored *out-of-core.*
- Highly efficient for GPU computing, or mobile computing (phones, etc).
- Consider the problem of computing the dominant k eigenvectors/eigenvalues of a dense matrix of size $m \times n$. Reduction in complexity from $O(mnk)$ to $O(mn \log k)$.
- Single pass algorithms have been developed for *streaming environments.*
- Randomized compression of *rank-structured matrices* ($\mathcal{H}$-matrices, discretized integral operators, inverse of discretized differential operators, etc).
- The randomization idea can be used to overcome a classical problem in numerical linear algebra: How to efficiently implement column-pivoted QR factorizations. (How to cast the computation as BLAS3 operations instead of BLAS2.)

A very fast *randomized* implementation of column pivoted QR



Speedup attained by a randomized algorithm 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 (SISC 2017). Closely related work by Duersch and Gu, SISC 2017 / SIREV 2020.*

References on randomized SVD:

- N. Halko, P. G. Martinsson, J. A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, 2011.
- P.G. Martinsson & J.A. Tropp, *Randomized Numerical Linear Algebra: Foundations & Algorithms*. 2020 Acta Numerica. arxiv #2002.01387.
- P.G. Martinsson, V. Rokhlin and M. Tygert (2006a), A randomized algorithm for the approximation of matrices, Technical Report Yale CS research report YALEU/DCS/RR-1361, Yale.
- Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert, *Randomized algorithms for the low-rank approximation of matrices*. PNAS 2007 104: 20167-20172.
- F. Woolfe, E. Liberty, V. Rokhlin, M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Applied and Computational Harmonic Analysis, **25**(3), 2008.
- V. Rokhlin, A. Szlam, and M. Tygert, *A Randomized Algorithm for Principal Component Analysis*, SIAM J. Matrix Anal. Appl., 31(3), 1100–1124, 2009.
- P.G. Martinsson, V. Rokhlin, and M. Tygert, *A randomized algorithm for the approximation of matrices*. Applied and Computational Harmonic Analysis, 30(1), pp. 47–68, 2011.

Relevant prior work in:

- C. H. Papadimitriou, P. Raghavan, H. Tamaki and S. Vempala (2000), *Latent semantic indexing: a probabilistic analysis*, Vol. 61, pp. 217–235.
- A. Frieze, R. Kannan and S. Vempala (2004), *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM 51(6), 1025–1041.

Outline:

- **Randomized algorithms for low rank approximation**

  Randomized singular value decomposition (RSVD) and randomized embeddings.

  Fast Johnson-Lindenstrauss transforms.

  Streaming and out-of-core algorithms.

- **Other examples randomized algorithms for linear algebraic problems**

  Randomization as a pre-conditioner in linear solvers.

  Sketching to reduce data storage in scientific simulations.

  Randomized sampling methods for otherwise inaccessible problems.

- **Randomized algorithms in high performance computing**

  Las Vegas vs. Monte Carlo. How RSVD fits ("in between" but closer to LV).

  Reliability and trustworthiness of the output. Certificates of accuracy.

  Reproducibility, impact on software development, etc.

Randomized linear solvers and randomized preconditioning

- **Overdetermined least squares problems.**

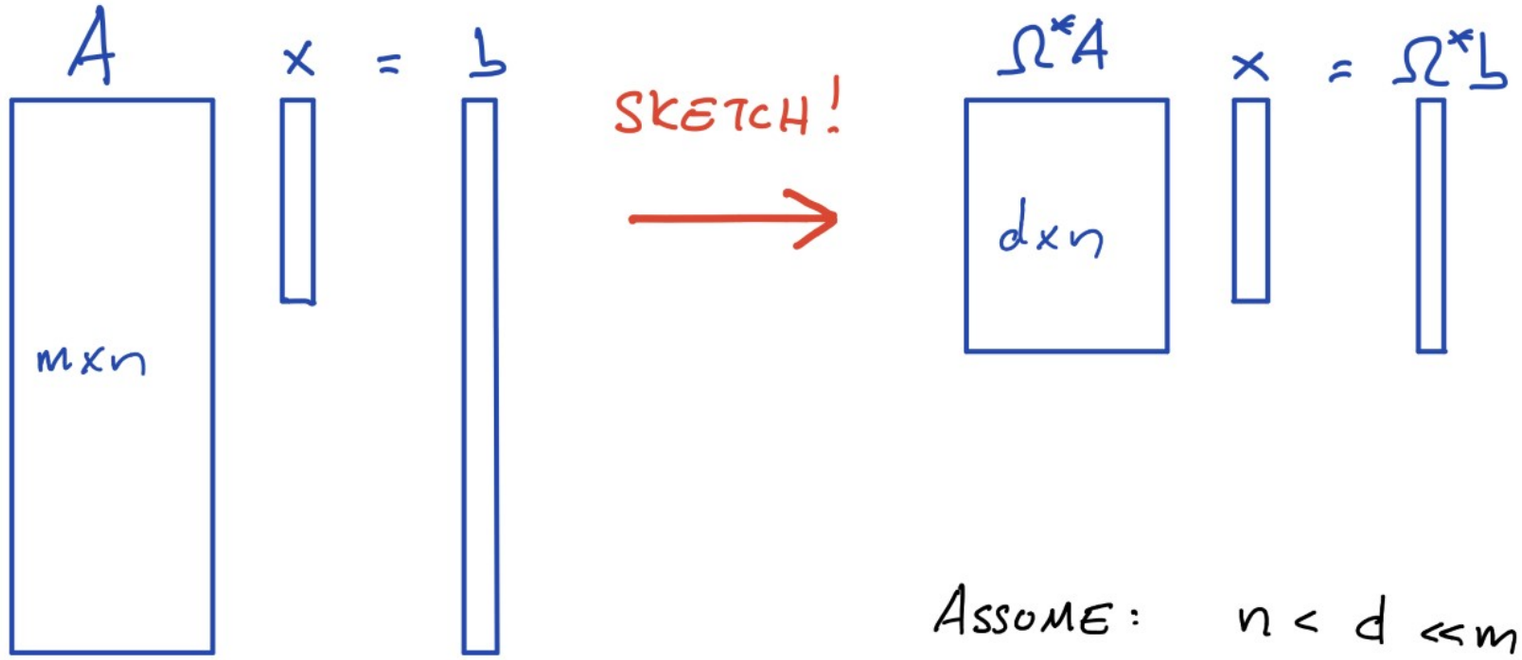
  Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$.

Randomized linear solvers and randomized preconditioning

- **Overdetermined least squares problems.**

  Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$.

  Draw a random embedding $\mathbf{\Omega} \in \mathbb{R}^{m \times d}$ and construct a smaller *sketched* system.



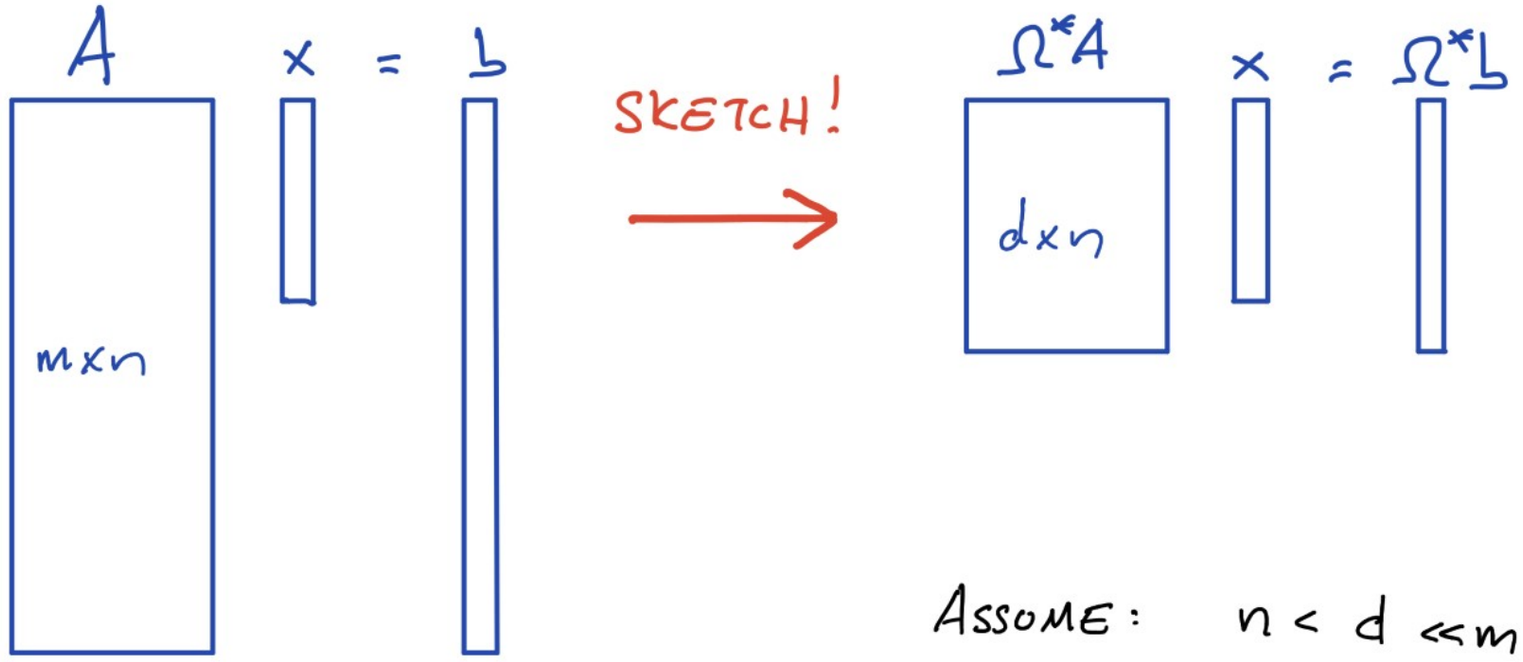
ASSUME: $n < d \ll m$

Randomized linear solvers and randomized preconditioning

- **Overdetermined least squares problems.**

  Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \gg n$, and that you seek to solve $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|$.

  Draw a random embedding $\mathbf{\Omega} \in \mathbb{R}^{m \times d}$ and construct a smaller *sketched* system.



A bold approach — "sketch-to-solve":

Find the vector $\mathbf{x}$ that solves the sketched system.

A safe approach — "sketch-to-precondition":

Build a *preconditioner* $\mathbf{M} \in \mathbb{R}^{n \times n}$ by factorizing $\mathbf{\Omega}^*\mathbf{A}$ so that $\mathbf{\Omega}^*\mathbf{A} = \mathbf{QM}$.

Iterate on the preconditioned linear system $(\mathbf{AM}^{-1})(\mathbf{Mx}) = \mathbf{b}$.

Rokhlin/Tygert (2008), Avron/Maymounkov/Toledo (2010), many more

Randomized linear solvers and randomized preconditioning

- **Overdetermined least squares problems.** *Sketch-to-precondition paradigm.*
- **Randomized Kaczmarz:** Randomization in a classical algorithm. Particularly effective when randomized embeddings are incorporated. *(Strohmer/Vershynnin 2009, Needell/Ward/Srebro 2014, Gower/Richtarik 2015, Liu/Wright 2016, . . .)*
- **Eliminating pivoting in Gaussian elimination:** D. Stott Parker showed in 1995 that you can eschew pivoting in Gaussian elimination if you first "scramble" the coefficient matrix through "pre-conditioning" via random unitary maps. Early example of fast J-L transform! Related work by Demmel/Dumitriu/Holtz (2007).
- **Graph Laplacians:** Linear systems whose coefficient matrix is a graph Laplacian can be solved using randomized methods in close to linear (in the number of edges) complexity. The idea is to compute an approximate Cholesky factorization $\mathbf{A} \approx \mathbf{C}\mathbf{C}^*$, and then use $\mathbf{C}$ as a preconditioner in conjugate gradients. Can be hybridized with ideas from nested dissection to achieve high practical speed for problems in scientific computing. *(Spielman/Teng 2004, Kyng/Sachdeva 2016, Koutis/Miller/Tolliver 2011, Livne/Brandt 2012, Spielman 2020, Chen/Liang/Biros 2020, . . .).*

Sketching in scientific computing

Storage of scientific data is often a core challenge:

- Massive sensor arrays.
- High resolution and multiband imaging.
- Time stepping in scientific simulations — fluid dynamics, molecular dynamics, etc.

In some environments, well-established techniques based on harmonic analysis (Fourier bases, wavelets, etc) are helpful.

When less is known about the data à priori, computing *randomized sketches* has proven to be valuable. Many different regimes have been pursued:

- Low rank matrices.
- Various tensor decompositions.
- Sparse Fourier transforms.

References: *Woolfe/Liberty/Rokhlin/Tygert (2008), Clarkson/Woodruff (2009), Halko/Martinsson/Tropp (2011), Li/Nguyen/Woodruff (2014), Ghashami/Liberty/Phillips/Woodruff (2016), Kressner/Periša (2017), Tropp/Yurtsever/Udell/Cevher (2017–), . . .*

Solution of huge linear systems

Randomized sampling has enabled the solution of some systems $\mathbf{Ax} = \mathbf{b}$ that would otherwise be intractable. Think of systems so large that even forming $\mathbf{A}$ is impossible.

In such situations, we must have some à priori information about structure or regularity in $\mathbf{A}$ that can be exploited. A common situation is that $\mathbf{A}(i,j) = k(\mathbf{x}_i, \mathbf{x}_j)$, where k is a given *kernel function*, and $\{\mathbf{x}_i\}_{i=1}^N$ is a given set of points in $\mathbb{R}^d$.

- **Kernel ridge regression:** Need to solve a linear system $\big(\mathbf{A} + \mu\mathbf{I}\big)\mathbf{x} = \mathbf{b}$ where $\mathbf{A}$ is a kernel matrix. Techniques based on Nyström approximation combined with randomized sampling have proven highly effective. *(Alaoui/Mahoney 2015, Avron/Clarkson/Woodruff 2017, Musco/Musco 2017, Rudi/Calandriello/Carratino/Rosasco 2018, . . .)*

- **Rank-structured representations of kernel matrices:** Multifrontal sparse direct solvers; FMMs; boundary integral equation methods; covariance matrices in Gaussian processes; log-determinants; etc. *(Martinsson 2011, March/Xiao/Biros 2015, Ambikasaran/Foreman-Mackey/Greengard/Hogg/O'Neil 2015, Ghysels/Li/Gorman/Rouet 2016, Yu/Levitt/Reiz/Biros 2017, Rebrova/Chávez/Liu/Ghysels/Li 2018, Geoga/Anitescu/Stein 2019, . . .)*

- **Electronic structure calculations:** Impossible even to store $\mathbf{x}$. *(Lim/Weare 2017)*

Probabilistic analysis of error accumulation

Aggregation of errors is becoming more and more of a problem as simulations get huge.

Single and half precision arithmetic exacerbate the situation. GPUs.

Worst case analysis of errors is no longer viable. Need probabilistic analysis instead.

Implications on Fast Multipole Methods, $\mathcal{H}$-matrix methods, etc.

Probability theory provides guidance in the design of algorithms and hardware systems.

Outline:

- **Randomized algorithms for low rank approximation**

  Randomized singular value decomposition (RSVD) and randomized embeddings.

  Fast Johnson-Lindenstrauss transforms.

  Streaming and out-of-core algorithms.

- **Other examples randomized algorithms for linear algebraic problems**

  Randomization as a pre-conditioner in linear solvers.

  Sketching to reduce data storage in scientific simulations.

  Randomized sampling methods for otherwise inaccessible problems.

- **Randomized algorithms in high performance computing**

  Las Vegas vs. Monte Carlo. How RSVD fits ("in between" but closer to LV).

  Reliability and trustworthiness of the output. Certificates of accuracy.

  Reproducibility, impact on software development, etc.

Nature of different randomized algorithms

Monte Carlo algorithms: Output is a random variable.

- Enable many otherwise intractable problems.
- Large variability in output, sensitive to poor random number generators, etc.
- Slow convergence. The error ε scales as $1/\sqrt{n}$ where n is number of instantiations. In consequence, $n \sim 1/\varepsilon^2$ number of samples required.

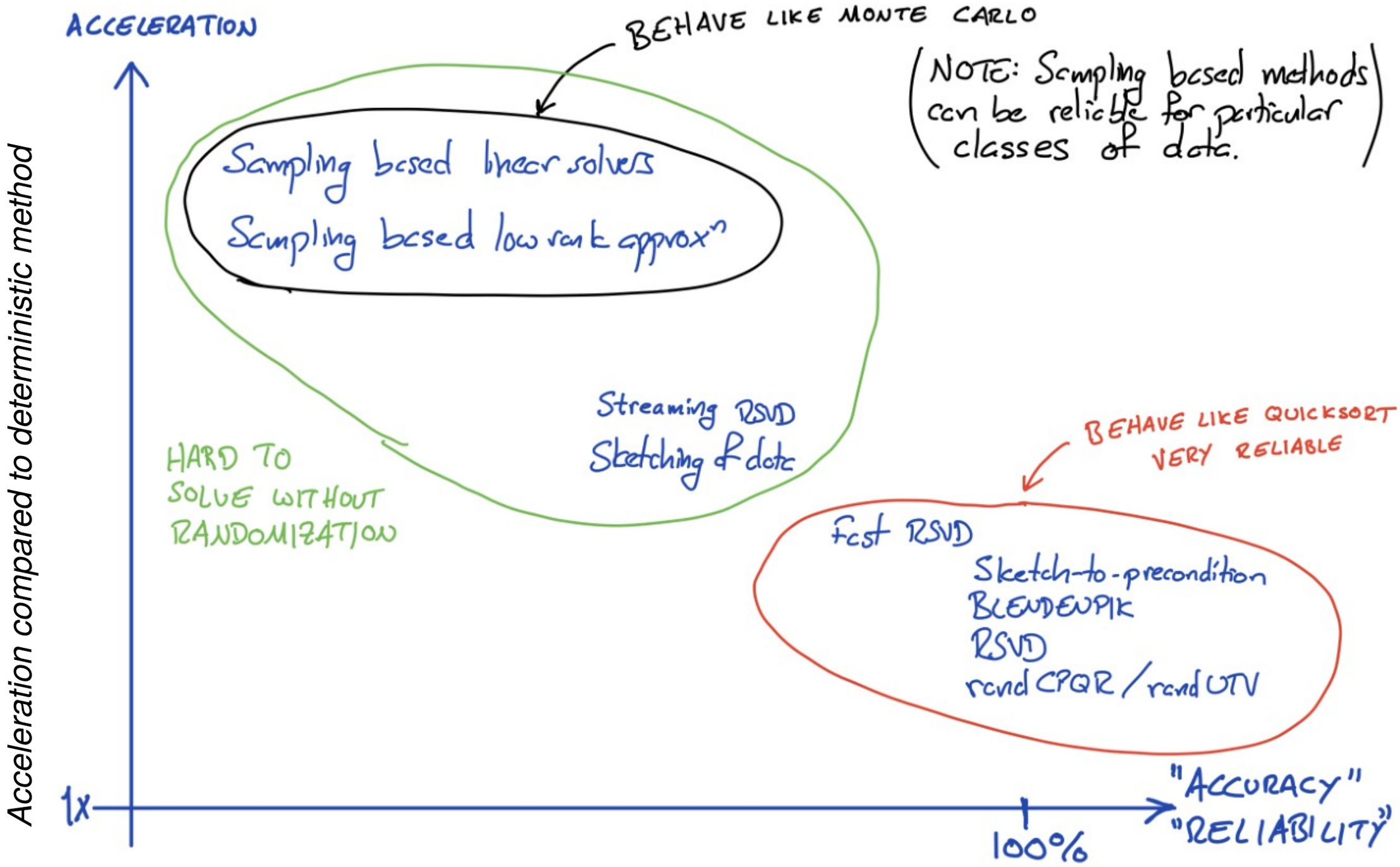
Las Vegas algorithms: Certain to find correct answer. Runtime is stochastic.

- Archetypical example is QuickSort.
- Randomized pre-conditioning is an LV algorithm in that the residual is controlled.

Many of the recently proposed randomized methods are "intermediate".

- Output is a random variable, but errors are very small and concentrated.
- Robust to quality of random number generators, etc.
- Large errors can be caught $\rightarrow$ only problem is occasionally excessive runtime.

Randomized algorithms come with different degrees of acceleration and reliability



Accuracy or reliability compared to deterministic method

This graphic is not to be taken *too* seriously…

The methods development pipeline

The methods development pipeline

- Need error estimators.
  Straightforward for linear solvers — just compute the residual.
  Can be built for other applications too. “Certificate of accuracy.”
- Need language for quantifying how variable the output is.
- Need to monitor the risk of unusual failures. E.g. when using fast J-L transforms.
- Need programming practices that can manage the extra unpredictability in how codes perform. E.g. by controlling the random seed, you attain reproducibility and easier code development.