Computing Rank Revealing Matrix Factorizations via Partial Pivoting

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, 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`

Note: An E-NLA seminar on March 10 elaborates on some themes raised here:

`https://www.youtube.com/watch?v=l262Qij6flM`

Research support by:

Objective: Compute a rank-revealing interpolatory decomposition

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ of approximate rank k, we seek to compute $\mathbf{C}$ and $\mathbf{Z}$ such that:

$$\begin{array}{cccc} \mathbf{A} & \approx & \mathbf{C} & \mathbf{Z}, \\ m\times n & & m\times k & k\times n \end{array}$$

where

(a) $\mathbf{C}$ holds a subset of the columns of $\mathbf{A}$, so that $\mathbf{C} = \mathbf{A}(:, J_{\rm s})$,

(b) $\|\mathbf{A} - \mathbf{C}(:, 1:i)\mathbf{C}(:, 1:i)^{\dagger}\mathbf{A}\| \approx \inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } i\}, \qquad \forall\, i \in \{1, 2, \dots, k\}$.

Objective: Compute a rank-revealing interpolatory decomposition

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ of approximate rank k, we seek to compute $\mathbf{C}$ and $\mathbf{Z}$ such that:

$$\begin{array}{cccc} \mathbf{A} & \approx & \mathbf{C} & \mathbf{Z}, \\ m\times n & & m\times k & k\times n \end{array}$$

where

(a) $\mathbf{C}$ holds a subset of the columns of $\mathbf{A}$, so that $\mathbf{C} = \mathbf{A}(:, J_{\rm s})$,

(b) $\|\mathbf{A} - \mathbf{C}(:, 1:i)\mathbf{C}(:, 1:i)^{\dagger}\mathbf{A}\| \approx \inf\{\|\mathbf{A} - \mathbf{B}\| \,:\, \mathbf{B} \text{ has rank } i\}, \qquad \forall\, i \in \{1, 2, \dots, k\}$.

Why insist on the interpolatory property (a)?

- The index vector $J_{\rm s}$ is useful in data interpretation.
- If $\mathbf{A}$ is sparse, then $\mathbf{C}$ is sparse.
- If $\mathbf{A}$ is non-negative, then $\mathbf{C}$ is non-negative.
- Storage efficient in that $\mathbf{C}$ sometimes does not need to be formed.
- Can be faster to compute than randomized SVD, and other competitors.

Objective: Compute a rank-revealing interpolatory decomposition

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ of approximate rank k, we seek to compute $\mathbf{C}$ and $\mathbf{Z}$ such that:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{C} & \mathbf{Z}, \\ m\times n & & m\times k & k\times n \end{array}$$

where

(a) $\mathbf{C}$ holds a subset of the columns of $\mathbf{A}$, so that $\mathbf{C} = \mathbf{A}(:, J_{\rm s})$,

(b) $\|\mathbf{A} - \mathbf{C}(:,1:i)\mathbf{C}(:,1:i)^{\dagger}\mathbf{A}\| \approx \inf\{\|\mathbf{A}-\mathbf{B}\| : \mathbf{B} \text{ has rank } i\}, \qquad \forall\, i \in \{1,2,\dots,k\}$.

Traditional algorithms:

- Column pivoted QR (“Gram Schmidt”).
- Fully pivoted LU (closely related to “ACA”).

Cost is $O(mnk)$, which is OK. But communication intensive $\rightarrow$ large prefactors.

Objective: Compute a rank-revealing interpolatory decomposition

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ of approximate rank k, we seek to compute $\mathbf{C}$ and $\mathbf{Z}$ such that:

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{C} & & \mathbf{Z}, \\ m\times n & & m\times k & & k\times n \end{array}$$

where

(a) $\mathbf{C}$ holds a subset of the columns of $\mathbf{A}$, so that $\mathbf{C} = \mathbf{A}(:,J_{\rm s})$,

(b) $\|\mathbf{A} - \mathbf{C}(:,1:i)\mathbf{C}(:,1:i)^{\dagger}\mathbf{A}\| \approx \inf\{\|\mathbf{A}-\mathbf{B}\| : \mathbf{B} \text{ has rank } i\}, \qquad \forall\, i \in \{1,2,\dots,k\}.$

Traditional algorithms:

- Column pivoted QR (“Gram Schmidt”).
- Fully pivoted LU (closely related to “ACA”).

Cost is $O(mnk)$, which is OK. But communication intensive $\rightarrow$ large prefactors.

Proposed randomized algorithms: Go through a “randomized sketch”.

- Faster practical speed in basically all cases.
- Improve on $O(mnk)$ complexity.
- Enables processing of large sparse matrices.
- Enables the use of *partially pivoted LU*. Much faster! (And interesting!)

Objective: Compute a rank-revealing interpolatory decomposition

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ of approximate rank k, we seek to compute $\mathbf{C}$ and $\mathbf{Z}$ such that:

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

where

(a) $\mathbf{C}$ holds a subset of the columns of $\mathbf{A}$, so that $\mathbf{C} = \mathbf{A}(:, J_{\rm s})$,

(b) $\|\mathbf{A} - \mathbf{C}(:, 1:i)\mathbf{C}(:, 1:i)^{\dagger}\mathbf{A}\| \approx \inf\{\|\mathbf{A}-\mathbf{B}\| \,:\, \mathbf{B} \text{ has rank } i\}, \qquad \forall\, i \in \{1, 2, \dots, k\}.$

Traditional algorithms:

- Column pivoted QR (“Gram Schmidt”).
- Fully pivoted LU (closely related to “ACA”).

Cost is $O(mnk)$, which is OK. But communication intensive $\rightarrow$ large prefactors.

Proposed randomized algorithms: Go through a “randomized sketch”.

- Faster practical speed in basically all cases.
- Improve on $O(mnk)$ complexity.
- Enables processing of large sparse matrices.
- Enables the use of *partially pivoted LU*. Much faster! (And interesting!)

Numerical experiments comparing different random embeddings will be presented.

The column selection problem

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector $J_{\rm s}$ of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}.$$

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

The column selection problem

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector $J_{\rm s}$ of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}.$$

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of $\mathbf{A}$. At cost $O(mnk)$, you get the factorization

$$\begin{array}{cccc} \mathbf{A}(:, J) & \approx & \mathbf{Q} & \left[\begin{array}{cc} \mathbf{R}_{11} & \mathbf{R}_{12} \end{array}\right] \\ m \times n & & m \times k & \begin{array}{cc} k \times k & k \times (n-k) \end{array} \end{array}$$

Set $J_{\rm s} = J(1:k)$, and pull out the factor $\mathbf{R}_{11}$ to form the interpolatory decomposition:

$$\mathbf{A}(:, J) \approx \mathbf{Q}\mathbf{R}_{11} \quad [\mathbf{I} \quad \mathbf{R}_{11}^{-1}\mathbf{R}_{12}] = \mathbf{A}(:, J_{\rm s})\,\mathbf{Z}(:, J).$$

The column selection problem

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector $J_{\rm s}$ of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}.$$

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of $\mathbf{A}$. At cost $O(mnk)$, you get the factorization

$$\begin{array}{cccc} \mathbf{A}(:, J) & \approx & \mathbf{Q} & [\ \mathbf{R}_{11} \quad \mathbf{R}_{12}\ ] \\ m \times n & & m \times k & \ k \times k \quad k \times (n-k) \end{array}$$

Set $J_{\rm s} = J(1:k)$, and pull out the factor $\mathbf{R}_{11}$ to form the interpolatory decomposition:

$$\mathbf{A}(:, J) \approx \mathbf{Q}\mathbf{R}_{11} \quad [\mathbf{I} \quad \mathbf{R}_{11}^{-1}\mathbf{R}_{12}] = \mathbf{A}(:, J_{\rm s})\,\mathbf{Z}(:, J).$$

Notes:

- Orthonormality must be maintained *scrupulously*. Use Householder or “double” Gram-Schmidt.
- CPQR can in principle fail (e.g. the “Kahan counter example”), but in practice it works very well.
- Reasonably computationally efficient for matrices that fit in RAM.
- Sophisticated versions of CPQR have been developed that *guarantee* close to optimal column selection, as well as bounding all elements of $\mathbf{R}_{11}^{-1}\mathbf{R}_{12}$. *(Gu & Eisenstat SISC 1996)*

The column selection problem

Problem formulation: Given an $m \times n$ matrix $\mathbf{A}$, and a target rank k, find an index vector $J_{\rm s}$ of length k, and a matrix $\mathbf{Z}$ of size $k \times n$ such that

$$\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}.$$

We additionally require $\mathbf{Z}$ to contain the $k \times k$ identity matrix as a submatrix.

A classical solution — column pivoted QR: Simply execute k steps of Gram-Schmidt orthogonalization on the columns of $\mathbf{A}$. At cost $O(mnk)$, you get the factorization

$$\begin{array}{cccc} \mathbf{A}(:,J) & \approx & \mathbf{Q} & \left[\begin{array}{cc} \mathbf{R}_{11} & \mathbf{R}_{12} \end{array}\right] \\ m \times n & & m \times k & \;\; k \times k \;\; k \times (n-k) \end{array}$$

Set $J_{\rm s} = J(1:k)$, and pull out the factor $\mathbf{R}_{11}$ to form the interpolatory decomposition:

$$\mathbf{A}(:,J) \approx \mathbf{Q}\mathbf{R}_{11}\;\;[\mathbf{I}\;\;\mathbf{R}_{11}^{-1}\mathbf{R}_{12}] = \mathbf{A}(:,J_{\rm s})\,\mathbf{Z}(:,J).$$

Questions:

- Can you efficiently process large matrices that do not fit in fast memory?
- Can you efficiently process huge *sparse* matrices?
- Can you improve on the practical speed of CPQR? Even on the $O(mnk)$ complexity?

The column selection problem — through a *sketch*

Simple theorem: Let **A** be an $m \times n$ matrix of exact rank k. Suppose:

(1) We have by some means computed a factorization

$$\underset{m\times n}{\mathbf{A}} \;=\; \underset{m\times k}{\mathbf{E}}\;\underset{k\times n}{\mathbf{F}}.$$

(2) We have solved the column selection problem for **F**, so that

$$\underset{k\times n}{\mathbf{F}} \;=\; \underset{k\times k}{\mathbf{F}(:,J_{\rm s})}\;\underset{k\times n}{\mathbf{Z}}.$$

Then, *automatically,* we have also solved the column selection problem for **A**:

$$\underset{m\times n}{\mathbf{A}} \;=\; \underset{m\times k}{\mathbf{A}(:,J_{\rm s})}\;\underset{k\times n}{\mathbf{Z}}.$$

The column selection problem — through a *sketch*

Simple theorem: Let **A** be an $m \times n$ matrix of exact rank k. Suppose:

(1) We have by some means computed a factorization

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}}.$$

(2) We have solved the column selection problem for **F**, so that

$$\underset{k \times n}{\mathbf{F}} = \underset{k \times k}{\mathbf{F}(:, J_{\rm s})} \; \underset{k \times n}{\mathbf{Z}}.$$

Then, *automatically,* we have also solved the column selection problem for **A**:

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{A}(:, J_{\rm s})} \; \underset{k \times n}{\mathbf{Z}}.$$

Proof: Assume that

(1) $$\mathbf{A} = \mathbf{E}\mathbf{F}$$

and that

(2) $$\mathbf{F} = \mathbf{F}(:, J_{\rm s})\mathbf{Z}.$$

Then

$$\mathbf{A}(:\,, J_{\rm s})\mathbf{Z} \overset{(2)}{=} \mathbf{E}\mathbf{F}(:\,, J_{\rm s})\mathbf{Z} \overset{(3)}{=} \mathbf{E}\mathbf{F} = \mathbf{A}.$$

The column selection problem — through a *sketch*

Simple theorem: Let **A** be an $m \times n$ matrix of exact rank k. Suppose:

(1) We have by some means computed a factorization

$$\begin{array}{cccc} \mathbf{A} & = & \mathbf{E} & \mathbf{F}. \\ m \times n & & m \times k & k \times n \end{array}$$

(2) We have solved the column selection problem for **F**, so that

$$\begin{array}{cccc} \mathbf{F} & = & \mathbf{F}(:, J_{\rm s}) & \mathbf{Z}. \\ k \times n & & k \times k & k \times n \end{array}$$

Then, *automatically,* we have also solved the column selection problem for **A**:

$$\begin{array}{cccc} \mathbf{A} & = & \mathbf{A}(:, J_{\rm s}) & \mathbf{Z}. \\ m \times n & & m \times k & k \times n \end{array}$$

Question: How do you find a $k \times n$ matrix **F** such that $\mathbf{A} = \mathbf{E}\mathbf{F}$ for some **E**?

The column selection problem — through a *sketch*

Simple theorem: Let **A** be an $m \times n$ matrix of exact rank k. Suppose:

(1) We have by some means computed a factorization

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{E}} \; \underset{k\times n}{\mathbf{F}}.$$

(2) We have solved the column selection problem for **F**, so that

$$\underset{k\times n}{\mathbf{F}} = \underset{k\times k}{\mathbf{F}(:,J_{\rm s})} \; \underset{k\times n}{\mathbf{Z}}.$$

Then, *automatically,* we have also solved the column selection problem for **A**:

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{A}(:,J_{\rm s})} \; \underset{k\times n}{\mathbf{Z}}.$$

Question: How do you find a $k \times n$ matrix **F** such that $\mathbf{A} = \mathbf{EF}$ for some **E**?

Randomized embedding! Draw a $k \times m$ Gaussian random matrix $\mathbf{\Omega}$ and set

$$\mathbf{F} = \mathbf{\Omega}\mathbf{A}.$$

The probability that $\mathbf{A} = \mathbf{EF}$ for some **E** is 1. (Of course, $\mathbf{E} = \mathbf{A}\mathbf{F}^{\dagger}$.)

*We do not need to know the factor **E**! It just never enters the computation.*

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector $J_{\rm s}$ and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}$.

```
(1)   Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;
(2)   Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
(3)   Do $k$ steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_{\rm s})\mathbf{Z}$.
```

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector $J_{\rm s}$ and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}$.

```
(1)  Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;
(2)  Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega A}$;
(3)  Do $k$ steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_{\rm s})\mathbf{Z}$.
```

Basic version: Use a *Gaussian* random matrix.

Complexity is $O(mnk)$ for a general dense matrix. Very high *practical* speed.

Complexity is $O(\mathrm{nnz}(\mathbf{A})k)$ for a sparse matrix. Again, high *practical* speed.

In some ways optimal sampling. Well supported by theory, e.g.:

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

(Skeletonization slightly increases the error: $\|\mathbf{A} - \mathbf{A}(:, J_{\rm s})\mathbf{Z}\| \ge \|\mathbf{A} - \mathbf{AF}^{\dagger}\mathbf{F}\|$.)

Note: The probability that a set of k columns is sampled is in a certain sense proportional to its spanning volume. This is precisely the property we are after.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector J_s and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_\mathrm{s})\mathbf{Z}$.

```
(1)  Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;
(2)  Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega A}$;
(3)  Do $k$ steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_\mathrm{s})\mathbf{Z}$.
```

Variation: Use a Gaussian matrix $\mathbf{\Omega}$, and incorporate *power iteration*.

Replace $\mathbf{F} = \mathbf{\Omega A}$ in step (2) by $\mathbf{F} = \mathbf{\Omega}(\mathbf{AA}^*)^q\mathbf{A}$ for a small integer q. Say $q = 1$ or $q = 2$. (I.e. do classical subspace iteration.)

This makes row($\mathbf{F}$) better aligned with the space spanned by the dominant k right singular vectors of $\mathbf{A}$.

Enhances accuracy, at cost of more work. Strong supporting theory. E.g., with $p = k$:

$$\mathbb{E}\left[\|\mathbf{A} - \mathbf{AF}^\dagger\mathbf{F}\|\right] \leq \left(1 + 4\sqrt{\frac{2\ \min(m,n)}{k-1}}\right)^{1/(2q+1)} \sigma_{k+1}(\mathbf{A}).$$

Re-orthonormalization is sometimes required to avoid loss of accuracy due to round-off.

Algorithm: Select spanning columns through a sketch

```
Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank $k$, and an over-sampling parameter $p$. (Say $p = 5$ or $p = 10$.)
Outputs: An index vector $J_s$ and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.
(1)   Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;
(2)   Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
(3)   Do $k$ steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.
```

Variation: Use a *structured random matrix* $\mathbf{\Omega}$. ("Fast Johnson-Lindenstrauss transform")

Idea: Use a matrix $\mathbf{\Omega}$ for which $\mathbf{\Omega}\mathbf{A}$ can be evaluated efficiently.

- Subsampled Randomized Fourier Transform (SRFT). Can be applied using FFT like methods. Cost is $O(mn\log(k))$ instead of $O(mnk)$.

- Sparse random matrix. Put only a couple of non-zero entries in each column. (Entries can be restricted to ± 1 for additional efficiency.) $O(mn)$ cost attainable.

Works quite well in practice. SRFTs are almost as good as Gaussians.
However, *far* weaker theory is available.

Cannot be combined with power iteration.

Ailon/Chazelle 2006; Liberty/Rokhlin/Tygert/Woolfe 2006; Halko/Martinsson/Tropp 2011; Clarkson/Woodruff 2013.

Algorithm: Select spanning columns through a sketch

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p. (Say $p = 5$ or $p = 10$.)

Outputs: An index vector $J_{\rm s}$ and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}$.

```
(1)   Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;
(2)   Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
(3)   Do $k$ steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_{\rm s})\mathbf{Z}$.
```

Variation: Discrete empirical interpolation method (DEIM)

Idea: Use the sample matrix $\mathbf{F}$ to build an approximate truncated SVD of $\mathbf{A}$.
(Form SVD of $(\mathbf{A}\mathbf{F}^{\dagger})\,\mathbf{F}$.) Requires one additional matrix-matrix multiplication.

Then perform partially pivoted LU on a thin matrix formed by the singular vectors to pick the spanning columns.

DEIM sometimes produces slightly more optimal column selection than CPQR.

DEIM can be faster than doing CPQR directly. (Which is perhaps counterintuitive!)

Often excellent choice.

Reference: Sorensen & Embree SISC 2016.

Algorithm: Select spanning columns through a sketch

```
Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank $k$, and an over-sampling parameter $p$. (Say $p = 5$ or $p = 10$.)
Outputs: An index vector $J_s$ and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_s)\mathbf{Z}$.
(1)  Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;
(2)  Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
(3)  Do $k$ steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_s)\mathbf{Z}$.
```

Variation: "Poor man's DEIM"

Idea: Skip forming the SVD, and just do partially pivoted LU on $\mathbf{F}^*$ directly!

Very economical. Works about as well as either CPQR, or regular DEIM.

Becomes particularly accurate with one step of power iteration.

À posteriori error analysis can be derived — monitor "growth factors" numerically. Work in progress.

Inspired by work by Trefethen and Schreiber (SIMAX 1990) on Gaussian elimination on random matrices.

```
Algorithm: Select spanning columns through a sketch
Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank $k$, and an over-sampling parameter $p$. (Say $p = 5$ or $p = 10$.)
Outputs: An index vector $J_{\rm s}$ and a $k \times n$ interpolation matrix $\mathbf{Z}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{Z}$.
(1)  Draw a $(k+p) \times m$ random matrix $\mathbf{\Omega}$;
(2)  Form a $(k+p) \times n$ matrix $\mathbf{F}$ holding samples from the row space, $\mathbf{F} = \mathbf{\Omega}\mathbf{A}$;
(3)  Do $k$ steps of Gram-Schmidt on the columns of $\mathbf{F}$ to form the factorization $\mathbf{F} \approx \mathbf{F}(:, J_{\rm s})\mathbf{Z}$.
```

Variation: Other possibilities:

- Can use sophisticated methods à la Gu-Eisenstat RRQR in Step (3). Leads to methods that are well supported by theory. Little improvement in practice, however.
- Can use the sketch to form an SVD. Then estimate *leverage scores*, and draw the columns through randomized sampling on the index set $\{1, 2, \dots, n\}$ using the resulting probability distribution. Rarely competitive in practice.
  (However, approaches of this type can be powerful for *huge* matrices where the matrix-vector multiplication is not accessible.)

Numerical experiments

Numerical experiments

Question: Which type of random matrix should I use for the sketching?

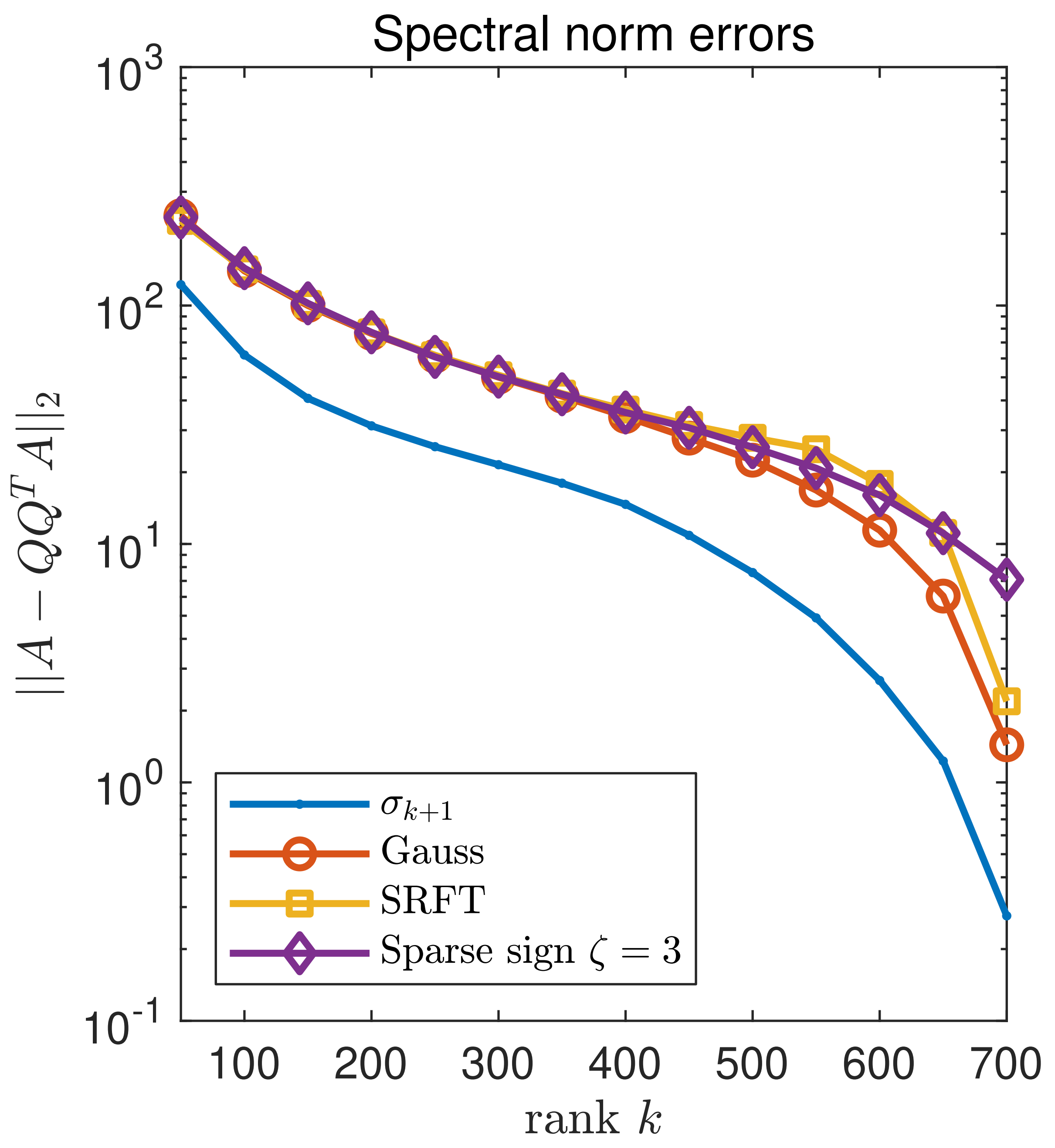
We will compare:

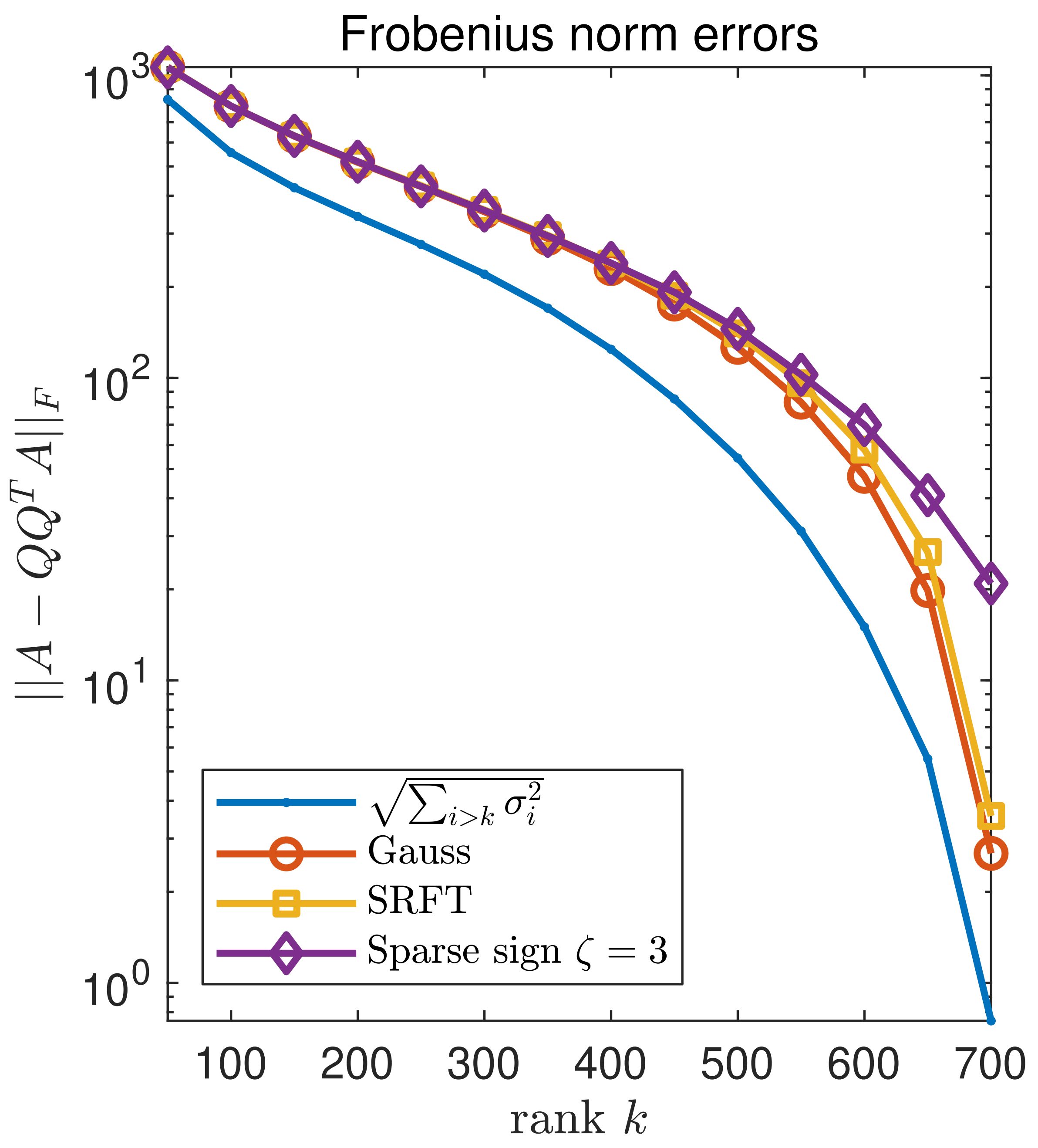
- Optimality: How good of a basis for the row space do you get?
- Computational cost: What is the *practical* speed?

Comparison of different random matrices — accuracy

Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

(To be precise, in these experiments, we sketched the *column space*, not the row space.)



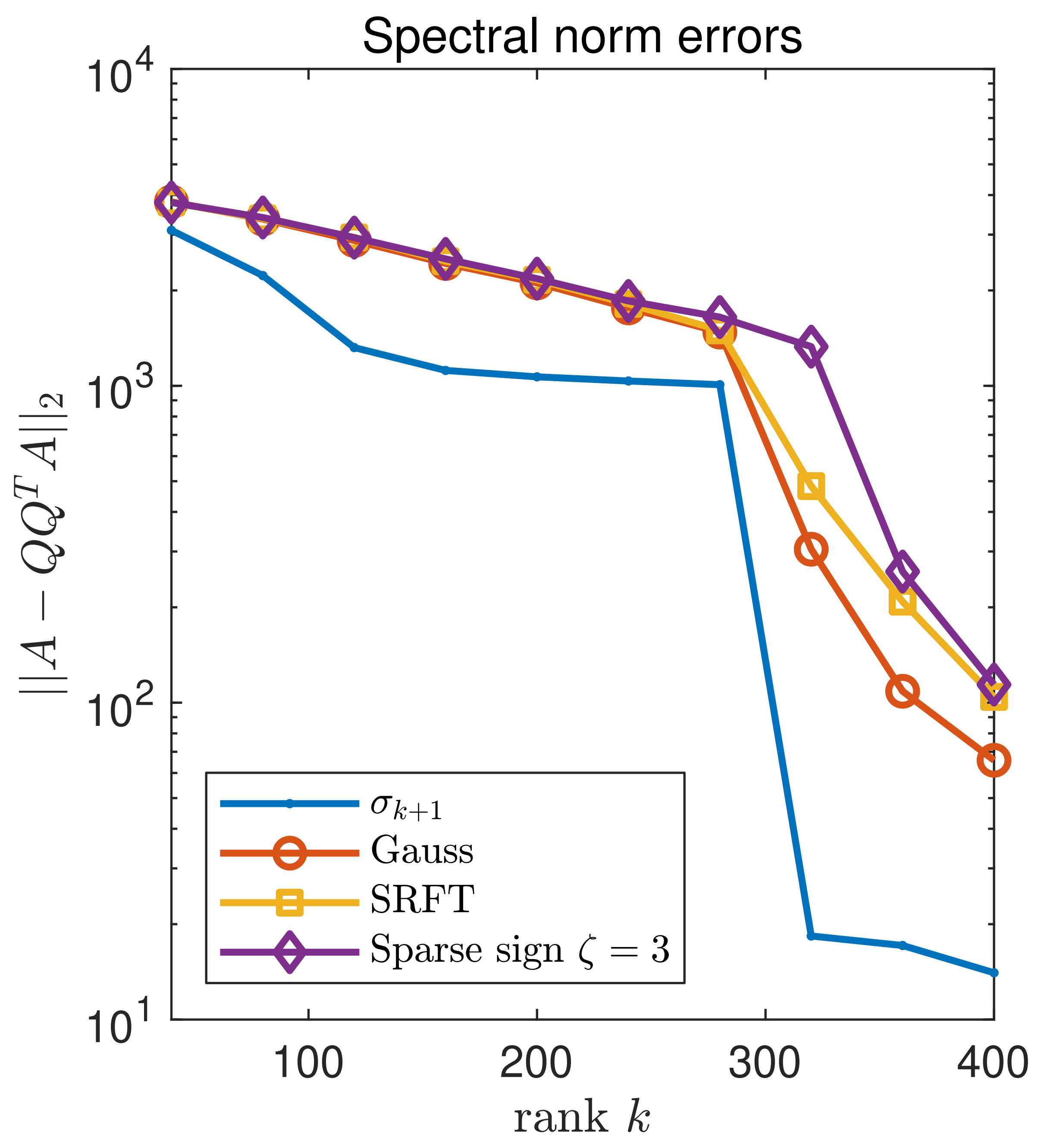


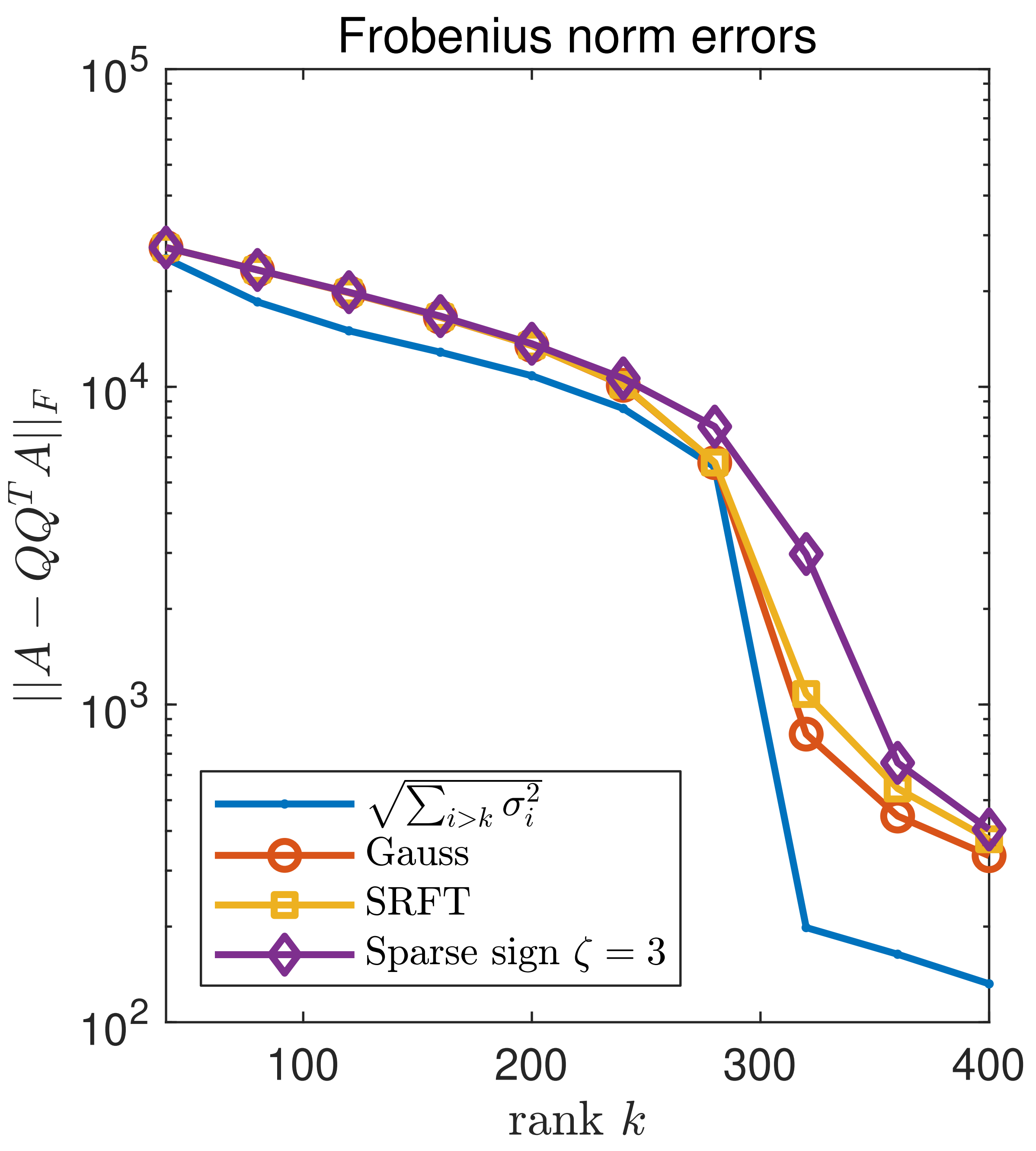
The "MNIST" test matrix is dense and of size $784 \times 60\,000$ *where each column holds one hard drawn digit between 0 and 9. The matrix is* 80% *sparse.*

Comparison of different random matrices — accuracy

Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

(To be precise, in these experiments, we sketched the *column space*, not the row space.)



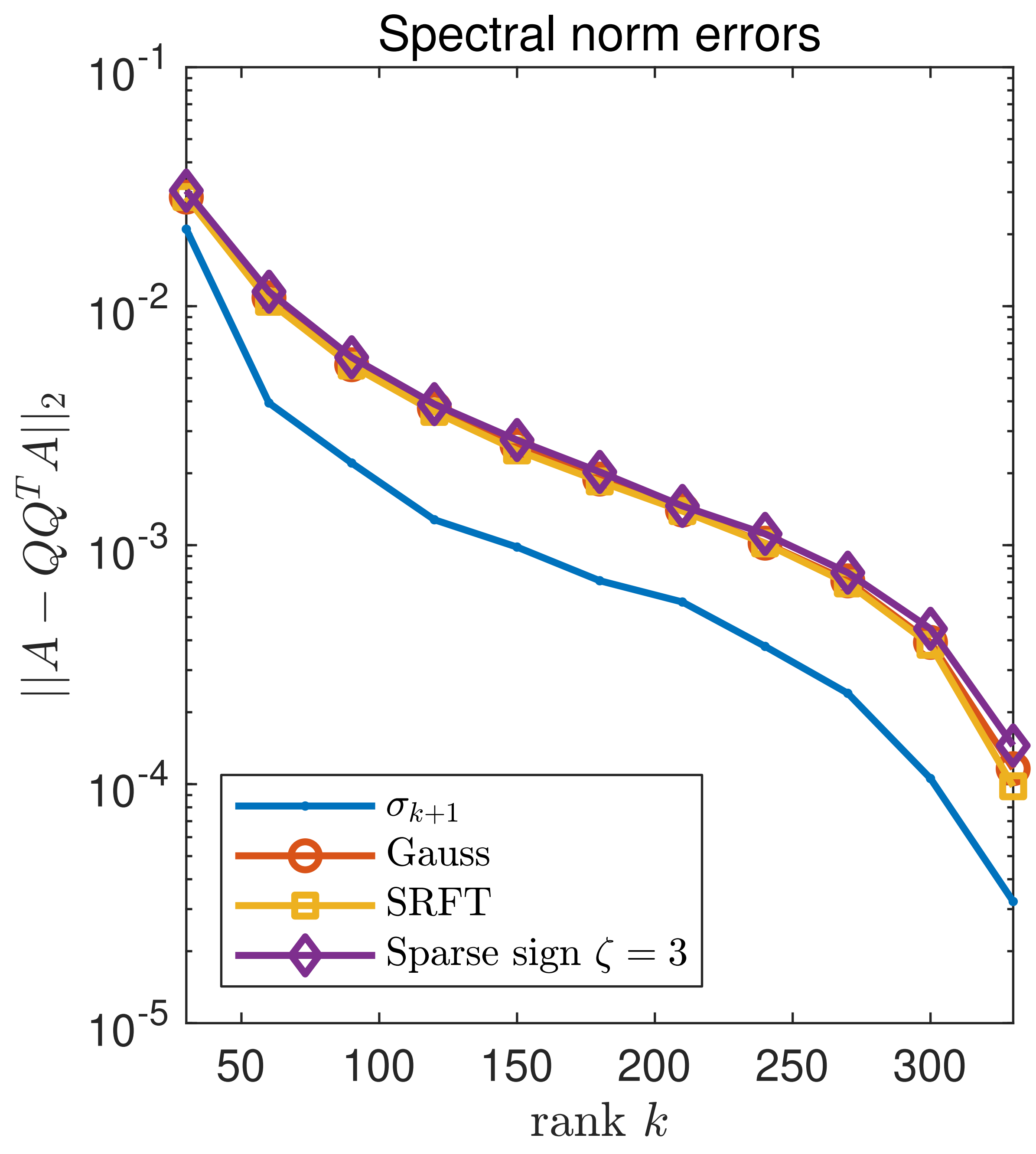


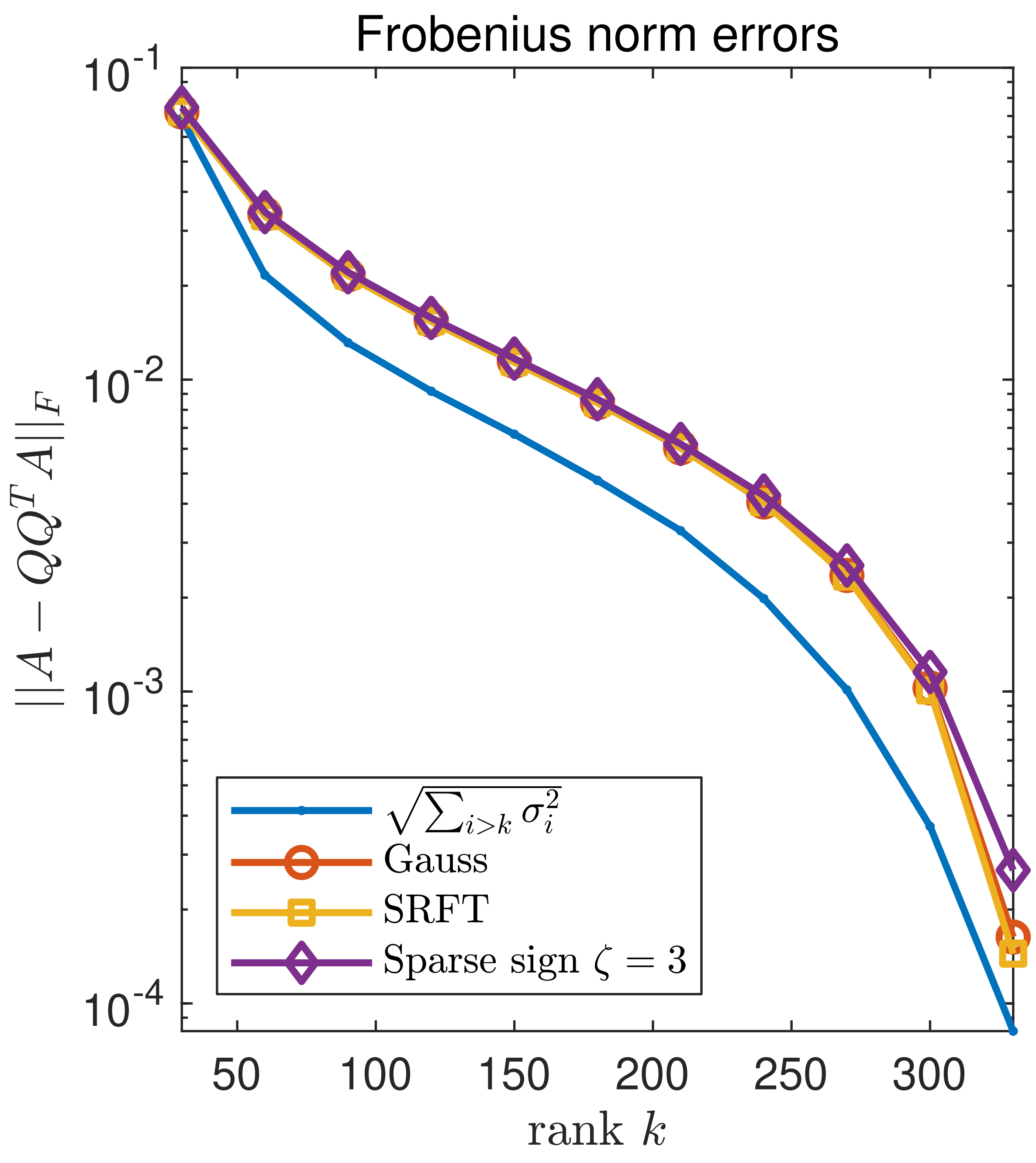
The "LARGE" test matrix is taken from a linear programming example. It is sparse, of size 4 282 $\times$ 8 617, *with* 20 635 *nonzero entries.*

Comparison of different random matrices — accuracy

Compare picking Ω as (1) Gaussian, (2) SRFT, (3) sparse random.

(To be precise, in these experiments, we sketched the *column space*, not the row space.)

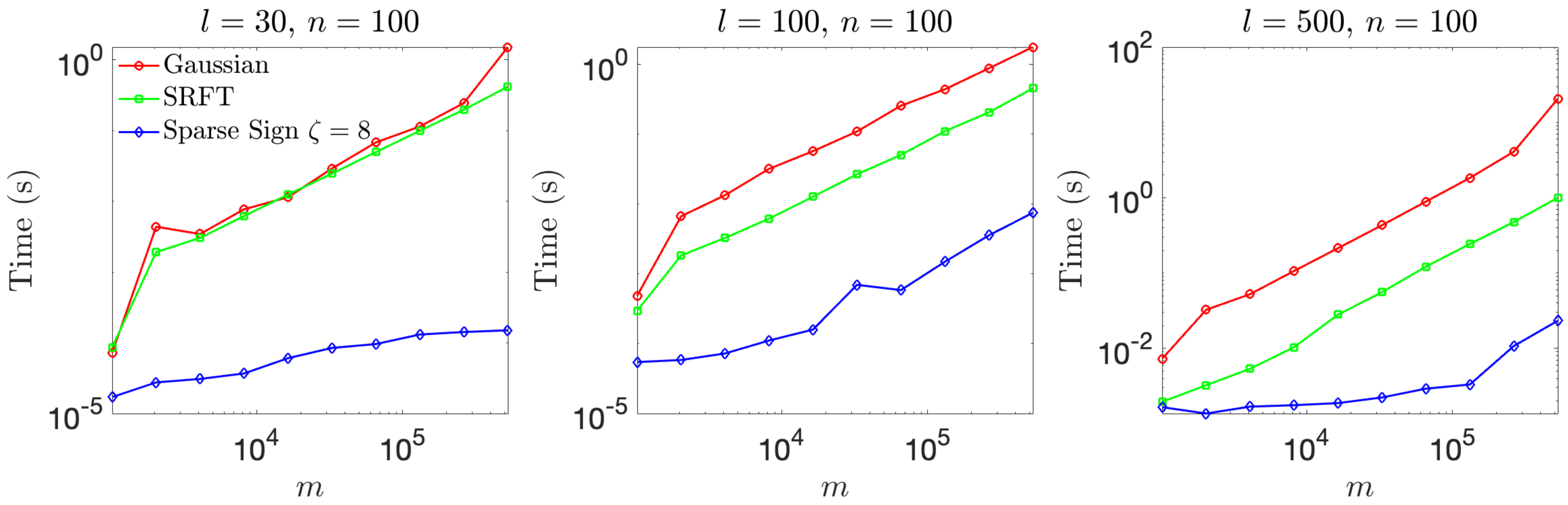




The "SNN" test matrix has been used in the CUR literature before. It is an artificial sparse matrix of size

$1\,000 \times 1\,000$.

Comparison of different random matrices — execution time



The runtime of applying different subspace embeddings $\boldsymbol{\Omega} \in \mathbb{R}^{\ell \times m}$ to an arbitrary dense matrix of size $m \times n$, scaled with respect to the ambient dimension m, at different embedding dimension l and a fixed number of columns $n = 100$.
(Note: This n is artificially small, but the scaling with n is linear.)

Note: Observe that the dimension of the sketch is quite high in these examples.

Numerical experiments

Question: How should I postprocess the matrix, once I have extracted a sketch?

We will compare:

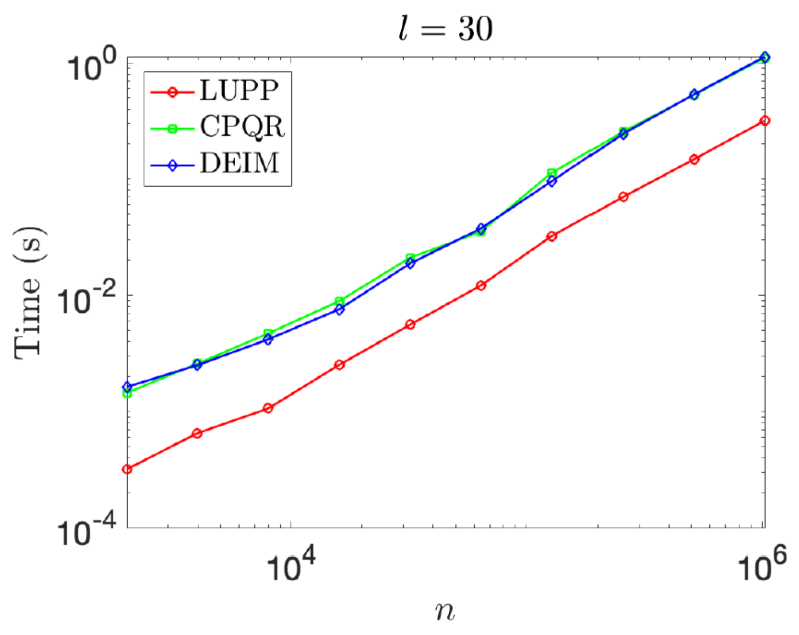
- Column pivoted QR
- DEIM: Form approximate SVD, then partially pivoted LU on the singular vectors.
- Partially pivoted LU directly on the sketching matrix. ("Poor man's DEIM")
- (Form approximate RSVD, then compute "leverage scores", then draw columns based on the leverage scores.)

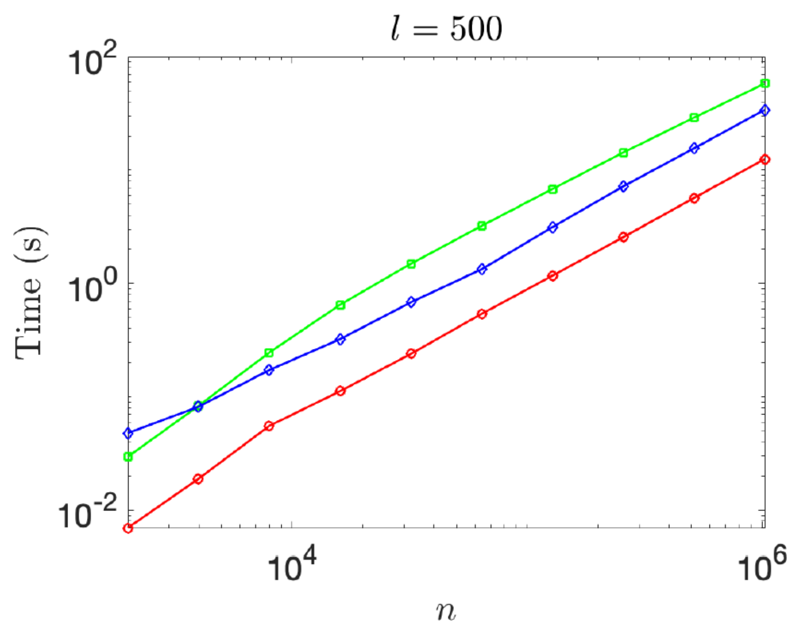
In these experiments, we use *Gaussian* random matrices.

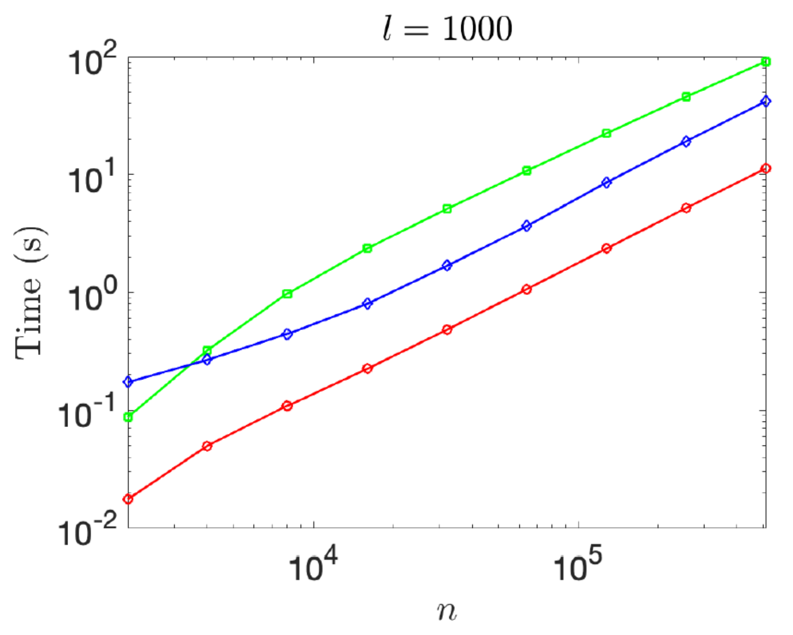
Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *runtime*.







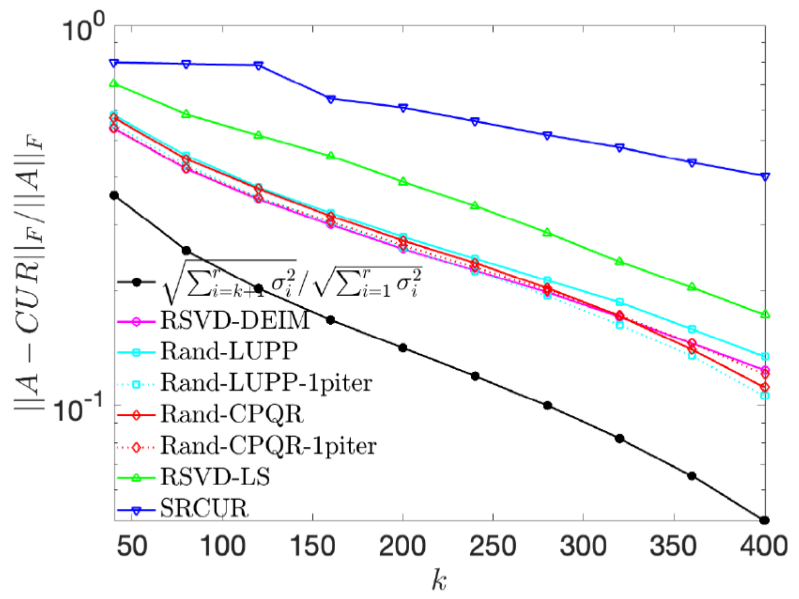
The runtime of various pivoting schemes on the sketches of size n $\times$ ℓ, scaled with respect to the problem size n, at different embedding dimension ℓ.

Observe that the dimension of the sketch is quite high in the last two examples.

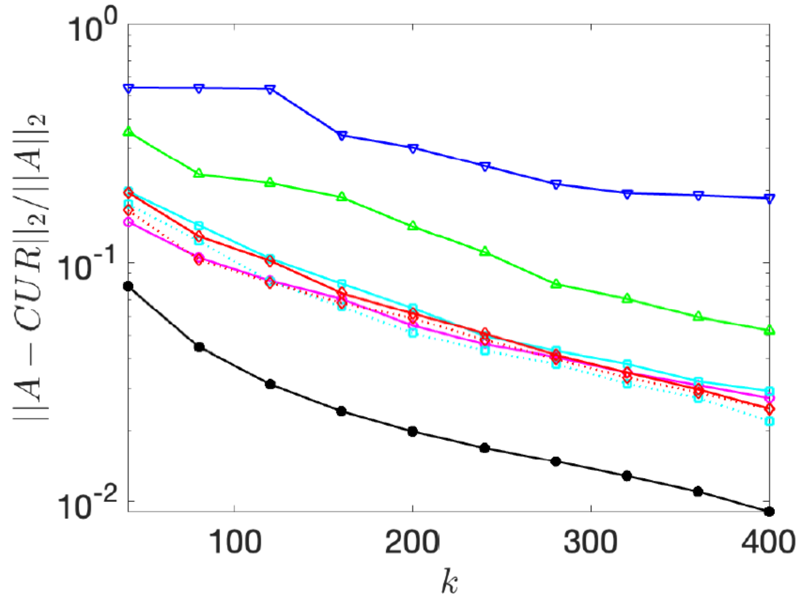
Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

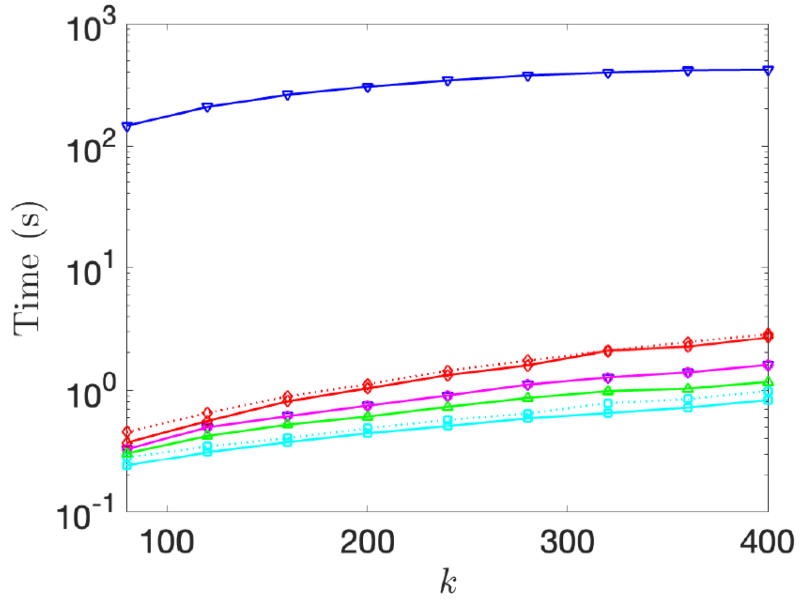
These are experiments to test the *accuracy / optimality*.



(a) Frobenius norm error.



(b) Spectral norm error.



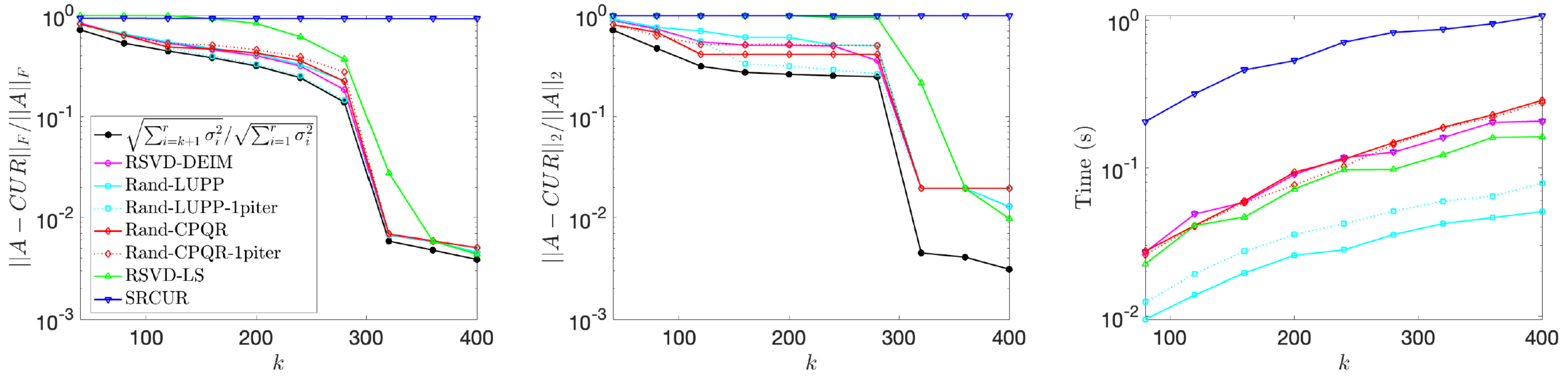
(c) Runtime.

The “MNIST” test matrix is dense and of size $784 \times 60\,000$ *where each column holds one hard drawn digit between 0 and 9. The matrix is* 80% *sparse.*

Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.



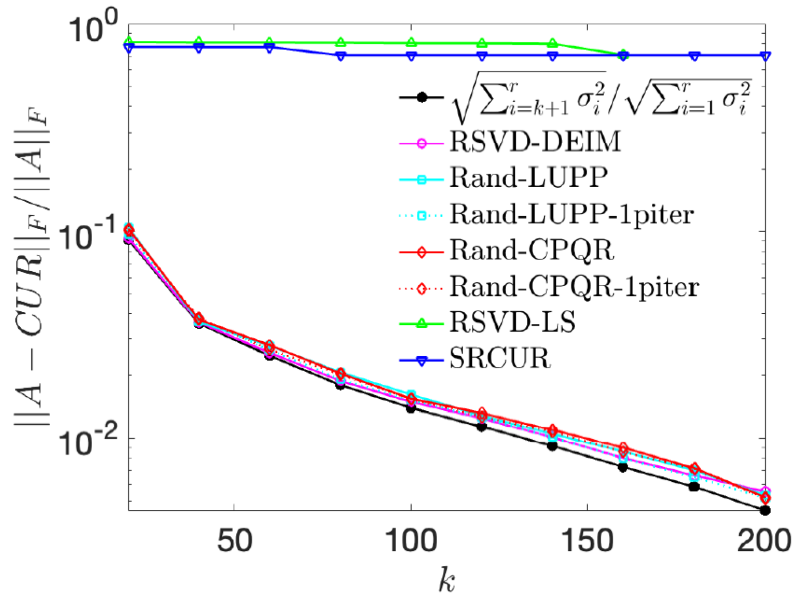
(a) Frobenius norm error. (b) Spectral norm error. (c) Runtime.

*The “*YALEFACE64X64*” test matrix holds 165 face images, each with* 64×64 *pixels. The pictures have been normalized, to create a dense matrix of size* 165×4096.

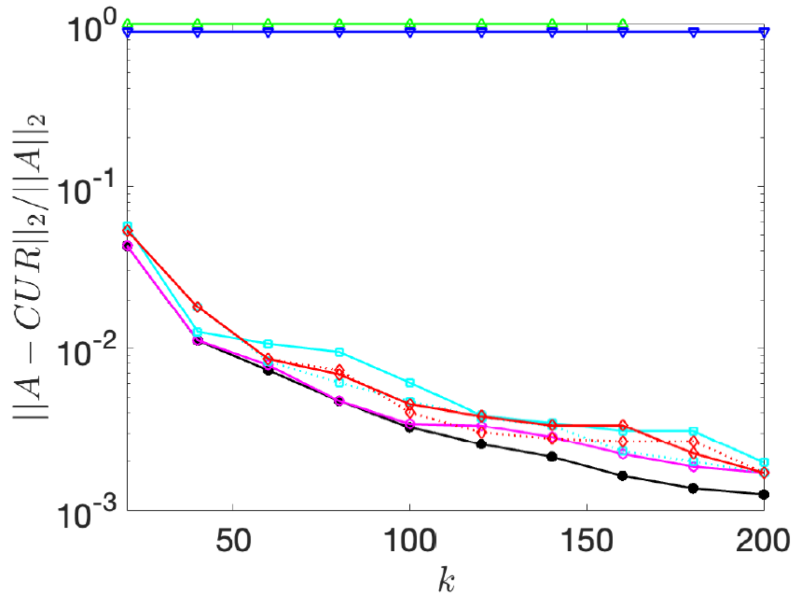
Comparison of different methods for solving the column selection problem

We compare different ways to “postprocess” the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

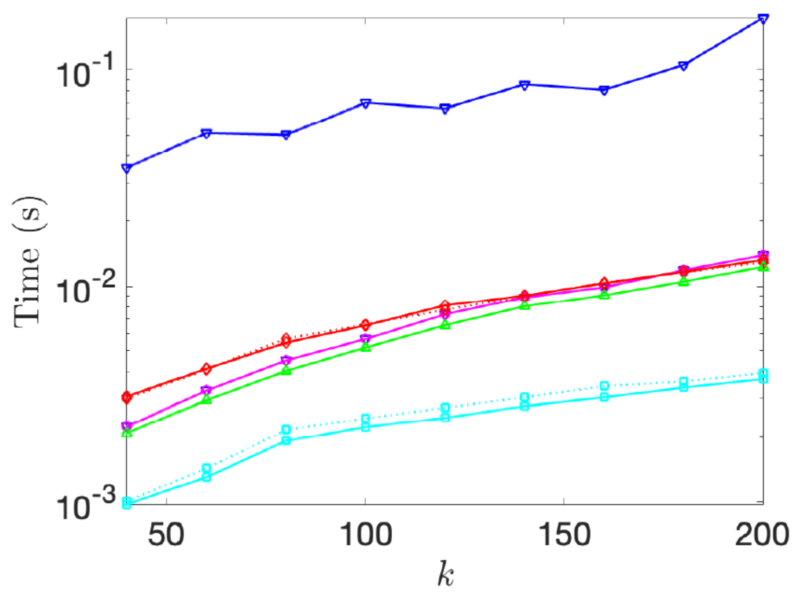
These are experiments to test the *accuracy / optimality*.



(a) Frobenius norm error.



(b) Spectral norm error.



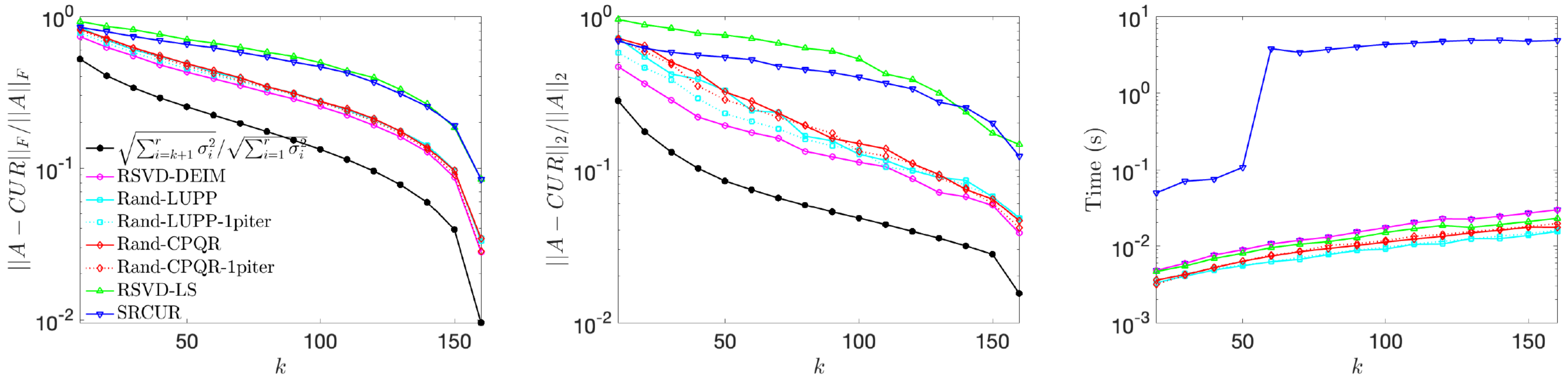
(c) Runtime.

The “SNN” test matrix has been used in the CUR literature before. It is an artificial sparse matrix of size $1\,000 \times 1\,000$.

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.

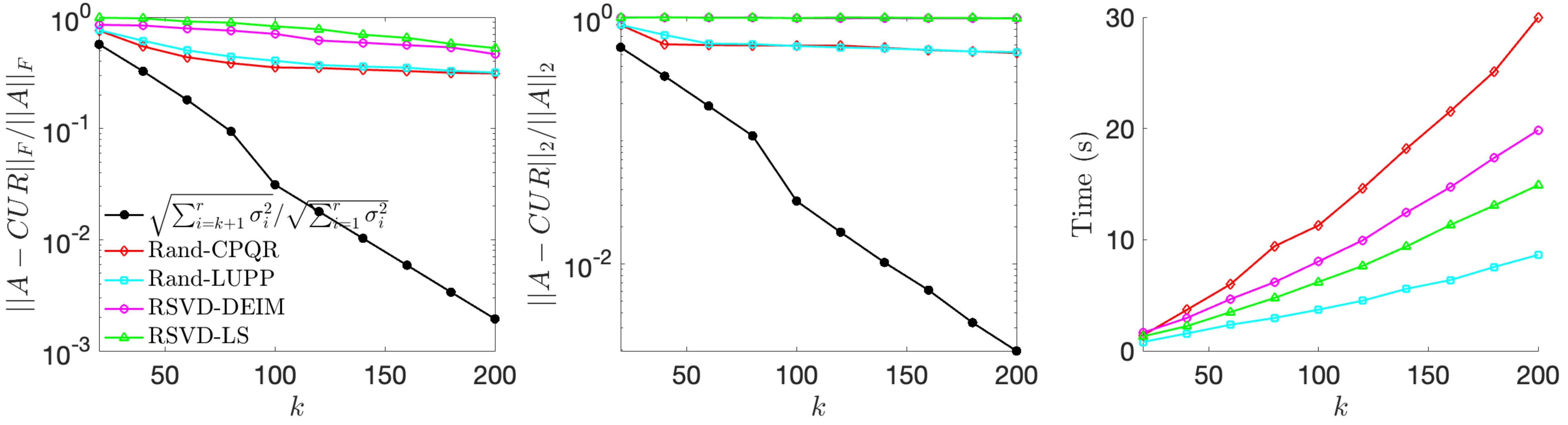


The "LARGE" test matrix is taken from a linear programming example. It is sparse, of size $4\,282 \times 8\,617$*, with* $20\,635$ *nonzero entries.*

Comparison of different methods for solving the column selection problem

We compare different ways to "postprocess" the extracted sketch $\mathbf{F} = \mathbf{\Omega A}$.

These are experiments to test the *accuracy / optimality*.



The "LARGESPARSE" test matrix is a synthetic non-negative matrix. It is sparse, of size $10^6 \times 10^6$.

Lessons from numerical experiments:

- Gaussian matrices are highly recommended. Excellent general purpose tools.
- "Sparse random" is *very fast* in all environments. Slightly less accurate.
- Subsampled trigonometric transforms are about as accurate as Gaussians.
  When the rank is large (say 500 or 1000), you see substantial speed gain.
- We tested three methods for picking columns from the sketch matrix:
  1. Column pivoted QR.
  2. DEIM. (Compute approximate RSVD, then do LU with partial pivoting.)
  3. Partially pivoted LU directly on the sketch. *It works because of randomization!!*

  They are about equally good at picking columns. DEIM perhaps slight winner.
  Partially pivoted LU ("Poor man's DEIM") is the fastest by a margin.

Lessons from numerical experiments:

- Gaussian matrices are highly recommended. Excellent general purpose tools.
- "Sparse random" is *very fast* in all environments. Slightly less accurate.
- Subsampled trigonometric transforms are about as accurate as Gaussians. When the rank is large (say 500 or 1000), you see substantial speed gain.
- We tested three methods for picking columns from the sketch matrix:
  1. Column pivoted QR.
  2. DEIM. (Compute approximate RSVD, then do LU with partial pivoting.)
  3. Partially pivoted LU directly on the sketch. *It works because of randomization!!*

  They are about equally good at picking columns. DEIM perhaps slight winner.
  Partially pivoted LU ("Poor man's DEIM") is the fastest by a margin.

Conclusion: Keep it simple!

Arxiv #2104.05877, with Yijun Dong of UT-Austin.

For another way to improve on speed of randomized SVD, see IP6 by Yuji Nakatsukasa.

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

Surveys:

- P.G. Martinsson and J. Tropp, "Randomized Numerical Linear Algebra: Foundations & Algorithms". *Acta Numerica*, 2020. Arxiv report 2002.01387.

  Long survey summarizing major findings in the field in the past decade.
- P.G. Martinsson, "Randomized methods for matrix computations." The Mathematics of Data, IAS/Park City Mathematics Series, 25(4), pp. 187 - 231, 2018.

  Book chapter that is written to be accessible to a broad audience. Focused on practical aspects rather than theory.
- N. Halko, P.G. Martinsson, J. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." *SIAM Review*, 53(2), 2011, pp. 217-288.

  Survey that describes the randomized SVD and its variations.

Tutorials, summer schools, etc:

- 2020: 3 lecture mini course on randomized linear algebra, KTH, Stockholm. Videos available.
- 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:

- ID: `http://tygert.com/software.html` (ID, SRFT, CPQR, etc)
- RSVDPACK: `https://github.com/sergeyvoronin` (RSVD, randomized ID and CUR)
- HQRRP: `https://github.com/flame/hqrrp/` (LAPACK compatible randomized CPQR)
- Randomized UTV: `https://github.com/flame/randutv`