

***PCMI Summer Session, The Mathematics of Data***

*Midway, Utah*

*July, 2016*

**Randomized algorithms for matrix computations and  
analysis of high dimensional data**

Gunnar Martinsson

The University of Colorado at Boulder

Slides and lecture notes at: [http://amath.colorado.edu/faculty/martinss/2016\\_PCMI/](http://amath.colorado.edu/faculty/martinss/2016_PCMI/)

(Or simply google “Gunnar Martinsson”)

*Research support by:*



## Objective:

Given an  $m \times n$  matrix  $\mathbf{A}$  we seek to compute a rank- $k$  approximation, typically with  $k \ll \min(m, n)$  (say  $m, n \sim 10^4, 10^6, 10^8, \dots$  and  $k \sim 10^2$  or  $10^3$ ),

$$\mathbf{A} \approx \mathbf{E} \mathbf{F}^* = \sum_{j=1}^k \mathbf{e}_j \mathbf{f}_j^*.$$

$m \times n \quad m \times k \quad k \times n$

Solving this problem leads to algorithms for computing:

- **Eigenvectors corresponding to leading eigenvalues.**  
(Require  $\mathbf{e}_j = \lambda_j \mathbf{f}_j$ , and  $\{\mathbf{f}_j\}_{j=1}^k$  to be orthonormal.)
- **Singular Value Decomposition (SVD) / Principal Component Analysis (PCA).**  
(Require  $\{\mathbf{e}_j\}_{j=1}^k$  and  $\{\mathbf{f}_j\}_{j=1}^k$  to be orthogonal sets.)
- **Spanning columns or rows.**  
(Require  $\{\mathbf{e}_j\}_{j=1}^k$  to be columns of  $\mathbf{A}$ , or require  $\{\mathbf{f}_j^*\}_{j=1}^k$  to be rows of  $\mathbf{A}$ .)
- *Etc*

The problem being addressed is ubiquitous in applications.

## Applications:

- Computational statistics (Principal Component Analysis, regression analysis, etc).
- Data mining, machine learning, analysis of network matrices, imaging, etc.  
(Algorithms related to PageRank, Latent Semantic Indexing, relaxed versions of k-means clustering, etc.)
- Accelerating standard packages for linear algebra.
- Nearest neighbor search for large clouds of points in high dimensional space.
- Diffusion geometry; a technique for constructing parameterizations on large collections of data points organized (modulo noise) along non-linear low-dimensional manifolds. Requires the computations of eigenvectors of *graph Laplace operators*.
- Fast algorithms for elliptic PDEs: more efficient Fast Multipole Methods, fast *direct* solvers, construction of special quadratures for corners and edges, etc.
- “General” pre-conditioners.
- Etc.

## Review of existing methods I

For a dense  $n \times n$  matrix that fits in RAM, excellent algorithms are already part of LAPACK (and incorporated into Matlab, Mathematica, *etc*).

- Double precision accuracy.
- Very stable.
- $O(n^3)$  asymptotic complexity. Reasonably small constants.
- Require extensive random access to the matrix.

When the target rank  $k$  is much smaller than  $n$ , there also exist  $O(n^2 k)$  methods with similar characteristics (the well-known Golub-Businger method, RRQR by Gu and Eisentstat, *etc*).

For small matrices, the state-of-the-art is quite satisfactory.

(By “small,” we mean something like  $n \leq 10\,000$  on today’s computers.)

**For kicks, we will improve on it anyway, but this is not the main point.**

## Review of existing methods II

If the matrix is large, but can rapidly be applied to a vector (if it is sparse, or sparse in Fourier space, or amenable to the FMM, etc.), so called *Krylov subspace methods* often yield excellent accuracy and speed.

The idea is to pick a starting vector  $\mathbf{r}$  (often a random vector), “restrict” the matrix  $\mathbf{A}$  to the  $k$ -dimensional “Krylov subspace”

$$\text{Span}(\mathbf{r}, \mathbf{A} \mathbf{r}, \mathbf{A}^2 \mathbf{r}, \dots, \mathbf{A}^{k-1} \mathbf{r})$$

and compute an eigendecomposition of the resulting matrix.

Advantages:

- Very simple access to  $\mathbf{A}$ .
- Extremely high accuracy possible. (Double precision accuracy for “converged” eigenmodes, etc.)

Drawbacks:

- The matrix is typically revisited  $O(k)$  times if a rank- $k$  approximation is sought. (Blocked versions exist, but the convergence analysis is less developed.)
- Numerical stability issues. These are well-studied and can be overcome, but they make software less portable (between applications, hardware platforms, etc.).

## “New” challenges in algorithmic design:

The existing state-of-the-art methods of numerical linear algebra that we have very briefly outlined were designed for an environment where the matrix fits in RAM and the key to performance was to minimize the number of *floating point operations* required.

Currently, *communication* is becoming the real bottleneck:

- While clock speed is hardly improving at all anymore, the cost of a flop keeps going down rapidly. (Multi-core processors, GPUs, cloud computing, etc.)
- The cost of slow storage (hard drives, flash memory, etc.) is also going down rapidly.
- Communication costs are decreasing, but *not* rapidly.
  - Moving data from a hard-drive.
  - Moving data between nodes of a parallel machine. (Or cloud computer ... )
  - The amount of fast cache memory close to a processor is not improving much.  
(In fact, it could be said to be *shrinking* — GPUs, multi-core, etc.)
- “Deluge of data”. Driven by ever cheaper storage and acquisition techniques. Web search, data mining in archives of documents or photos, hyper-spectral imagery, social networks, gene arrays, proteomics data, sensor networks, financial transactions, ...

The more powerful computing machinery becomes,  
the more important efficient algorithm design becomes.

- Linear scaling (w.r.t. problem size, processors, etc.).
- Minimal data movement.

That *randomization* can be used to overcome some of the communication bottlenecks in matrix computations has been pointed out by several authors:

C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala (2000)

A. Frieze, R. Kannan, and S. Vempala (1999, 2004)

D. Achlioptas and F. McSherry (2001)

P. Drineas, R. Kannan, M. W. Mahoney, and S. Muthukrishnan (2006)

S. Har-Peled (2006)

A. Deshpande and S. Vempala (2006)

S. Friedland, M. Kaveh, A. Niknejad, and H. Zare (2006)

T. Sarlós (2006a, 2006b, 2006c)

M. Rudelson, R. Vershynin (2007)

K. Clarkson, D. Woodruff (2009)

... deluge of papers ...

*Covered by other lecturers in this summer school. Lectures of P. Drineas in particular.*

*Literature surveys: Halko, Martinsson, Tropp (2011). Mahoney (2011). Woodruff (2014). Etc.*



## *Review of existing methods III*

Examples of how randomization could be used:

- **Random column/row selection**

Draw at random some columns and suppose that they span the entire column space.

If rows are drawn as well, then spectral properties can be estimated.

Crude sampling leads to less than  $O(mn)$  complexity, but is “dangerous.”

- **Sparsification**

Zero out the vast majority of the entries of the matrix. Keep a random subset of entries, and boost their magnitude to preserve “something.”

- **Quantization and sparsification**

Restrict the entries of the matrix to a small set of values (-1/0/1 for instance).

The methods outlined can be as fast as you like, but must necessarily have weak performance guarantees. They can work well for certain classes of matrices for which additional information is available (basically, matrices that are in some sense “over-sampled”).

## Approach advocated here:

A set of randomized algorithms for computing a rank- $k$  approximation to a matrix.

These methods are engineered from the ground up to:

- Minimize communication.
- Handle streaming data, or data stored “out-of-core.”
- Easily adapt to a broad range of distributed computing architectures.

Computational profile (for an  $m \times n$  matrix of approximate rank  $k$ ):

- At least  $O(mn)$  complexity. (To be precise:  $O(mnk)$  or  $O(mn \log k)$ .)

- The accuracy  $\varepsilon$  is a user-set number.

(If the application permits, it could be  $\varepsilon = 10^{-12}$  or less.)

- Since the method is randomized, it has a *failure probability*  $\eta$ .

$\eta$  is a user specified number.

The cost of the method grows as  $\eta \rightarrow 0$ , but setting  $\eta = 10^{-10}$  is cheap.

*For all practical purposes, the method succeeds with probability 1.*

- The only error that will be controlled is the “backwards error”  $\|\mathbf{A} - \mathbf{EF}\| \leq \text{TOL}$ .

**Goal:** Given an  $m \times n$  matrix  $\mathbf{A}$ , compute an approximate rank- $k$  SVD  $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$ .

---

*Brief review of some key facts of the SVD:*

Let  $\mathbf{A}$  be an  $m \times n$  matrix. Then  $\mathbf{A}$  admits a *singular value decomposition (SVD)*

$$\begin{array}{ccccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{D} & \mathbf{V}^*, \\ m \times n & & m \times r & r \times r & r \times n \end{array}$$

where  $r = \min(m, n)$  and where

$$\begin{array}{ll} \mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_r] & \text{is a matrix holding the "left singular vectors" } \mathbf{u}_i, \\ \mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \cdots \ \mathbf{v}_r] & \text{is a matrix holding the "right singular vectors" } \mathbf{v}_i, \\ \mathbf{D} = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r) & \text{is a diagonal matrix holding the "singular values" } \sigma_i. \end{array}$$

For any  $k$  such that  $1 \leq k \leq \min(m, n)$ , we define the *truncated SVD* as

$$\mathbf{A}_k = \mathbf{U}(:, 1 : k) \mathbf{D}(1 : k, 1 : k) \mathbf{V}(:, 1 : k)^* = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^*.$$

The following theorem states that  $\mathbf{A}_k$  is the "optimal" rank- $k$  approximation to  $\mathbf{A}$ :

**Theorem (Eckart-Young):** Let  $\|\cdot\|$  denote either the Frobenius or spectral norm. Then

$$\|\mathbf{A} - \mathbf{A}_k\| = \min\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ is of rank } k\}.$$

Moreover,

$$\|\mathbf{A} - \mathbf{A}_k\| = \sigma_{k+1},$$

when the spectral norm is used,

$$\|\mathbf{A} - \mathbf{A}_k\| = \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_r^2},$$

when the Frobenius norm is used.

**Goal:** Given an  $m \times n$  matrix  $\mathbf{A}$ , compute an approximate rank- $k$  SVD  $\mathbf{A} \approx \mathbf{UDV}^*$ .

---

*Brief review of some key facts of the SVD:* Recall the SVD

$$\begin{array}{ccccccc} \mathbf{A} & = & \mathbf{U} & \mathbf{D} & \mathbf{V}^* & = & \sum_{j=1}^r \sigma_j \mathbf{u}_j \mathbf{v}_j^* \\ m \times n & & m \times r & r \times r & r \times n & & \end{array}$$

where  $r = \min(m, n)$ . Some facts:

- The left singular vectors  $\{\mathbf{u}_j\}_{j=1}^k$  form an optimal basis for the column space of  $\mathbf{A}$  in the sense that  $\|\mathbf{A} - \mathbf{U}(:, 1:k)\mathbf{U}(:, 1:k)^* \mathbf{A}\| = \inf\{\|\mathbf{A} - \mathbf{PA}\| : \text{where } \mathbf{P} \text{ is an ON proj. to a } k\text{-dimensional space}\}$ .
- The right singular vectors  $\{\mathbf{v}_j\}_{j=1}^k$  form an optimal basis for the row space of  $\mathbf{A}$ .
- For a *symmetric* matrix, the eigenvalue decomposition (EVD) and the singular value decomposition are in many ways equivalent, and a truncated EVD is also an optimal rank- $k$  approximation.
- The EVD and the SVD are also in many ways equivalent for a *normal* matrix (recall that  $\mathbf{A}$  is normal if  $\mathbf{AA}^* = \mathbf{A}^* \mathbf{A}$ ), but the EVD might be complex even when  $\mathbf{A}$  is real.
- For *non-normal* matrices, eigenvectors and eigenvalues are generally not convenient tools for low rank approximation.
- For a general matrix, the SVD provides the EVDs of  $\mathbf{A}^* \mathbf{A}$  and  $\mathbf{AA}^*$ :

$$\mathbf{AA}^* = \mathbf{UD}^2\mathbf{U}^*, \quad \text{and} \quad \mathbf{A}^* \mathbf{A} = \mathbf{VD}^2\mathbf{V}^*.$$

The SVD can be derived from the relations above.

**Goal:** Given an  $m \times n$  matrix  $\mathbf{A}$ , compute an approximate rank- $k$  SVD  $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$ .

**Algorithm:**

1. *Find an  $m \times k$  orthonormal matrix  $\mathbf{Q}$  such that  $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$ .*
2. *(I.e., the columns of  $\mathbf{Q}$  form an ON-basis for the range of  $\mathbf{A}$ .)*
- 3.
4. Form the  $k \times n$  matrix  $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ .
5. Compute the SVD of the small matrix  $\mathbf{B}$  so that  $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$ .
6. Form the matrix  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ .

**Note:** Steps 4 – 6 are exact; the error in the method is all in  $\mathbf{Q}$ :

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

**Note:** The classical *Golub-Businger algorithm* follows this pattern. It finds  $\mathbf{Q}$  in Step 3 via direct orthogonalization of the columns of  $\mathbf{A}$  via, e.g., Gram-Schmidt.

**Range finding problem:** Given an  $m \times n$  matrix  $\mathbf{A}$  and an integer  $k < \min(m, n)$ , find an orthonormal  $m \times k$  matrix  $\mathbf{Q}$  such that  $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^* \mathbf{A}$ .

### Solving the primitive problem via randomized sampling — intuition:

1. Draw **random vectors**  $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k \in \mathbb{R}^n$ .  
(We will discuss the choice of distribution later — think Gaussian for now.)
2. Form “**sample**” vectors  $\mathbf{y}_1 = \mathbf{A}\mathbf{r}_1, \mathbf{y}_2 = \mathbf{A}\mathbf{r}_2, \dots, \mathbf{y}_k = \mathbf{A}\mathbf{r}_k \in \mathbb{R}^m$ .
3. Form **orthonormal vectors**  $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_k \in \mathbb{R}^m$  such that
$$\text{Span}(\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_k) = \text{Span}(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_k).$$

For instance, Gram-Schmidt can be used — pivoting is rarely required.

If  $\mathbf{A}$  has *exact* rank  $k$ , then  $\text{Span}\{\mathbf{q}_j\}_{j=1}^k = \text{Ran}(\mathbf{A})$  with probability 1.

What is perhaps surprising is that even in the general case,  $\{\mathbf{q}_j\}_{j=1}^k$  often does almost as good of a job as the theoretically optimal vectors (which happen to be the  $k$  leading left singular vectors).

**Range finding problem:** Given an  $m \times n$  matrix  $\mathbf{A}$  and an integer  $k < \min(m, n)$ , find an orthonormal  $m \times k$  matrix  $\mathbf{Q}$  such that  $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^* \mathbf{A}$ .

**Solving the primitive problem via randomized sampling — intuition:**

1. Draw a **random matrix**  $\mathbf{R} \in \mathbb{R}^{n \times k}$ .

(We will discuss the choice of distribution later — think Gaussian for now.)

2. Form a “**sample**” matrix  $\mathbf{Y} = \mathbf{A}\mathbf{R} \in \mathbb{R}^{m \times k}$ .

3. Form an **orthonormal matrix**  $\mathbf{Q} \in \mathbb{R}^{m \times k}$  such that  $\mathbf{Y} = \mathbf{Q}\mathbf{R}$ .

For instance, Gram-Schmidt can be used — pivoting is rarely required.

If  $\mathbf{A}$  has *exact* rank  $k$ , then  $\mathbf{A} = \mathbf{Q}\mathbf{Q}^* \mathbf{A}$  with probability 1.

**Goal:** Given an  $m \times n$  matrix  $\mathbf{A}$ , compute an approximate rank- $k$  SVD  $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$ .

**Algorithm:**

1. Draw an  $n \times k$  Gaussian random matrix  $\mathbf{R}$ .
2. Form the  $m \times k$  sample matrix  $\mathbf{Y} = \mathbf{A}\mathbf{R}$ .
3. Form an  $m \times k$  orthonormal matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{Q}\mathbf{R}$ .
4. Form the  $k \times n$  matrix  $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ .
5. Compute the SVD of the small matrix  $\mathbf{B}$ :  $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$ .
6. Form the matrix  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ .



**Goal:** Given an  $m \times n$  matrix  $\mathbf{A}$ , compute an approximate rank- $k$  SVD  $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$ .

### Algorithm:

1. Draw an  $n \times k$  Gaussian random matrix  $\mathbf{R}$ .  $\mathbf{R} = \text{randn}(n, k)$
2. Form the  $m \times k$  sample matrix  $\mathbf{Y} = \mathbf{A}\mathbf{R}$ .  $\mathbf{Y} = \mathbf{A} * \mathbf{R}$
3. Form an  $m \times k$  orthonormal matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{Q}\mathbf{R}$ .  $[\mathbf{Q}, \mathbf{R}] = \text{qr}(\mathbf{Y})$
4. Form the  $k \times n$  matrix  $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ .  $\mathbf{B} = \mathbf{Q}' * \mathbf{A}$
5. Compute the SVD of the small matrix  $\mathbf{B}$ :  $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$ .  $[\mathbf{Uhat}, \mathbf{Sigma}, \mathbf{V}] = \text{svd}(\mathbf{B}, 0)$
6. Form the matrix  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ .  $\mathbf{U} = \mathbf{Q} * \mathbf{Uhat}$

**Observation:** The proposed method interacts with  $\mathbf{A}$  exactly twice:

- The matrix-matrix multiplication on line 2:  $\mathbf{Y} = \mathbf{A}\mathbf{R}$ .
- The matrix-matrix multiplication on line 4:  $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ .

The matrix-matrix multiplication is a very efficient operation. It executes well on many different computing platforms — singlecore CPU, multicore CPU, distributed memory parallel machines, cloud computers. *Very fast on GPUs.*

Next, we will demonstrate that one can actually avoid the second visit to  $\mathbf{A}$ .

This allows us to process matrices so large they cannot be stored at all.

## Single pass algorithms

Consider a randomized algorithm for computing the EVD of a *symmetric* matrix  $\mathbf{A}$ :

*Input:* An  $n \times n$  symmetric matrix  $\mathbf{A}$ , and a target rank  $k$ .

*Output:* Rank- $k$  factors  $\mathbf{U}$  and  $\mathbf{D}$  that form an approximate EVD  $\mathbf{A} \approx \mathbf{UDU}^*$

1. Draw an  $n \times k$  Gaussian random matrix  $\mathbf{G}$ .
2. Compute an  $n \times k$  sample matrix  $\mathbf{Y} = \mathbf{AG}$ .
3. Compute an  $n \times k$  ON matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$ .
4. Project  $\mathbf{A}$  down to  $\mathbf{C} = \mathbf{Q}^*\mathbf{AQ}$ .
5. Compute the EVD of  $\mathbf{C}$  (which is small):  $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*$ .
6. Map back to original space  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ .

We see that  $\mathbf{A}$  is visited twice, in the computations high-lighted in red.

## Single pass algorithms

Consider a randomized algorithm for computing the EVD of a *symmetric* matrix  $\mathbf{A}$ :

*Input:* An  $n \times n$  symmetric matrix  $\mathbf{A}$ , and a target rank  $k$ .

*Output:* Rank- $k$  factors  $\mathbf{U}$  and  $\mathbf{D}$  that form an approximate EVD  $\mathbf{A} \approx \mathbf{UDU}^*$

1. Draw an  $n \times k$  Gaussian random matrix  $\mathbf{G}$ .
2. Compute an  $n \times k$  sample matrix  $\mathbf{Y} = \mathbf{AG}$ .
3. Compute an  $n \times k$  ON matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$ .

We claimed earlier that the columns of  $\mathbf{Q}$  approximately span  $\text{col}(\mathbf{A})$ , so

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

Since  $\mathbf{A}$  is symmetric, we also have  $\mathbf{A} \approx \mathbf{AQQ}^*$ , so

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{AQQ}^* = \mathbf{QCQ}^*,$$

where

$$\mathbf{C} := \mathbf{Q}^*\mathbf{AQ}.$$

Right multiply the definition of  $\mathbf{C}$  by  $\mathbf{Q}^*\mathbf{G}$  to get

$$\mathbf{CQ}^*\mathbf{G} = \mathbf{Q}^*\mathbf{AQQ}^*\mathbf{G} \approx \{\text{Use that } \mathbf{AQQ}^* \approx \mathbf{A}\} \approx \mathbf{Q}^*\mathbf{AG} = \mathbf{Q}^*\mathbf{Y}.$$

Observe that the quantities in red are known and can be formed inexpensively.

As a consequence, we can determine  $\mathbf{C}$  by solving the matrix equation:

$$\begin{array}{ccc} \mathbf{C} & (\mathbf{Q}^*\mathbf{G}) & = & (\mathbf{Q}^*\mathbf{Y}). \\ k \times k & k \times k & & k \times k \end{array}$$

A single pass algorithm for computing an approximate EVD of a **symmetric** matrix:

**Input:** An  $n \times n$  symmetric matrix  $\mathbf{A}$ , and a target rank  $k$ .

**Output:** Rank- $k$  factors  $\mathbf{U}$  and  $\mathbf{D}$  that form an approximate EVD  $\mathbf{A} \approx \mathbf{UDU}^*$

1. Draw an  $n \times k$  Gaussian random matrix  $\mathbf{G}$ .
2. Compute an  $n \times k$  sample matrix  $\mathbf{Y} = \mathbf{AG}$ .
3. Compute an  $n \times k$  ON matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$ .
4. Solve the matrix equation  $\mathbf{C}(\mathbf{Q}^*\mathbf{Y}) = (\mathbf{Q}^*\mathbf{G})$  for  $\mathbf{C}$ , enforcing  $\mathbf{C} = \mathbf{C}^*$ .
5. Compute the EVD of  $\mathbf{C}$  (which is small):  $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*$ .
6. Map back to original space  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ .

## Single pass algorithms

Now consider a *general*  $m \times n$  matrix  $\mathbf{A}$ . (Not necessarily symmetric.)

*Input:* An  $m \times n$  matrix  $\mathbf{A}$ , and a target rank  $k$ .

*Output:* Rank- $k$  factors  $\mathbf{U}$ ,  $\mathbf{D}$ , and  $\mathbf{V}$  that form an approximate SVD  $\mathbf{A} \approx \mathbf{UDV}^*$

1. ...?

2. ...?

3. ...?

4. ...?

5. ...?

6. ...?

Our old approach started:

1. Draw an  $n \times k$  Gaussian random matrix  $\mathbf{G}$ .
2. Compute an  $n \times k$  sample matrix  $\mathbf{Y} = \mathbf{AG}$ .
3. Compute an  $n \times k$  ON matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{QQ}^*\mathbf{Y}$ .

While the information in  $\mathbf{G}$  and  $\mathbf{Y}$  contains everything we need for a symmetric matrix, this simply is not true for a general matrix.

*We have no idea about the directions of the right singular vectors!*

## Single pass algorithms

Now consider a *general*  $m \times n$  matrix  $\mathbf{A}$ . (Not necessarily symmetric.)

*Input:* An  $m \times n$  matrix  $\mathbf{A}$ , and a target rank  $k$ .

*Output:* Rank- $k$  factors  $\mathbf{U}$ ,  $\mathbf{D}$ , and  $\mathbf{V}$  that form an approximate SVD  $\mathbf{A} \approx \mathbf{UDV}^*$

1. Draw two Gaussian random matrices  $\mathbf{G}_c$  of size  $n \times k$  and  $\mathbf{G}_r$  of size  $m \times k$ .
2. Form two sampling matrices  $\mathbf{Y}_c = \mathbf{AG}_c$  and  $\mathbf{Y}_r = \mathbf{A}^*\mathbf{G}_r$ . *This can be done in one pass!!*
3. Compute two basis matrices  $\mathbf{Q}_c = \text{orth}(\mathbf{Y}_c)$  and  $\mathbf{Q}_r = \text{orth}(\mathbf{Y}_r)$ .

The columns of  $\mathbf{Q}_c$  and  $\mathbf{Q}_r$  form approximate bases for the column and row spaces of  $\mathbf{A}$ , respectively, so

$$\mathbf{A} \approx \mathbf{Q}_c \mathbf{Q}_c^* \mathbf{A} \mathbf{Q}_r \mathbf{Q}_r^* = \mathbf{Q}_c \mathbf{C} \mathbf{Q}_r^*,$$

where

$$(1) \quad \mathbf{C} := \mathbf{Q}_c^* \mathbf{A} \mathbf{Q}_r.$$

First right multiply (1) by  $\mathbf{Q}_r^* \mathbf{G}_c$  to obtain

$$(2) \quad \mathbf{C} \mathbf{Q}_r^* \mathbf{G}_c = \mathbf{Q}_c^* \mathbf{A} \mathbf{Q}_r \mathbf{Q}_r^* \mathbf{G}_c \approx \{\text{Use that } \mathbf{A} \mathbf{Q}_r \mathbf{Q}_r^* \approx \mathbf{A}\} \approx \mathbf{Q}_c^* \mathbf{A} \mathbf{G}_c = \mathbf{Q}_c^* \mathbf{Y}_c.$$

Next left multiply (1) by  $\mathbf{G}_r^* \mathbf{Q}_c$  to obtain

$$(3) \quad \mathbf{G}_r^* \mathbf{Q}_c \mathbf{C} = \mathbf{G}_r^* \mathbf{Q}_c \mathbf{Q}_c^* \mathbf{A} \mathbf{Q}_r \approx \{\text{Use that } \mathbf{Q}_c \mathbf{Q}_c^* \mathbf{A} \approx \mathbf{A}\} \approx \mathbf{G}_r^* \mathbf{A} \mathbf{Q}_r = \mathbf{Y}_r^* \mathbf{Q}_r.$$

Finally, define  $\mathbf{C}$  as the least-square solution of the two equations

$$(\mathbf{G}_r^* \mathbf{Q}_c) \mathbf{C} = (\mathbf{Y}_r^* \mathbf{Q}_r) \quad \text{and} \quad \mathbf{C} (\mathbf{Q}_r^* \mathbf{G}_c) = (\mathbf{Q}_c^* \mathbf{Y}_c).$$

A single pass algorithm for computing an approximate SVD of a *general* matrix:

*Input:* An  $m \times n$  matrix  $\mathbf{A}$ , and a target rank  $k$ .

*Output:* Rank- $k$  factors  $\mathbf{U}$ ,  $\mathbf{D}$ , and  $\mathbf{V}$  that form an approximate SVD  $\mathbf{A} \approx \mathbf{UDV}^*$

1. Draw two Gaussian random matrices  $\mathbf{G}_c$  of size  $n \times k$  and  $\mathbf{G}_r$  of size  $m \times k$ .
2. Form two sampling matrices  $\mathbf{Y}_c = \mathbf{AG}_c$  and  $\mathbf{Y}_r = \mathbf{A}^*\mathbf{G}_r$ .



*Step 2 can be executed in one pass over the matrix.*

3. Compute two basis matrices  $\mathbf{Q}_c = \text{orth}(\mathbf{Y}_c)$  and  $\mathbf{Q}_r = \text{orth}(\mathbf{Y}_r)$ .
4. Determine  $\mathbf{C}$  by solving  $(\mathbf{G}_r^*\mathbf{Q}_c)\mathbf{C} = (\mathbf{Y}_r^*\mathbf{Q}_r)$  and  $\mathbf{C}(\mathbf{Q}_r^*\mathbf{G}_c) = (\mathbf{Q}_c^*\mathbf{Y}_c)$  for  $\mathbf{C}$ .
5. Compute the SVD of  $\mathbf{C}$  (which is small):  $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$ .
6. Map back to original space  $\mathbf{U} = \mathbf{Q}_c\hat{\mathbf{U}}$  and  $\mathbf{V} = \mathbf{Q}_r\hat{\mathbf{V}}$ .

## Single pass (“streaming”) algorithms:

<b>A is symmetric:</b>	<b>A is not symmetric:</b>
Generate a random matrix $\mathbf{G}$ .	Generate random matrices $\mathbf{G}_c$ and $\mathbf{G}_r$ .
Compute a sample matrix $\mathbf{Y}$ .	Compute sample matrices $\mathbf{Y}_c = \mathbf{A} \mathbf{G}_c$ and $\mathbf{Y}_r = \mathbf{A}^* \mathbf{G}_r$ .
Find an ON matrix $\mathbf{Q}$ such that $\mathbf{Y} = \mathbf{Q} \mathbf{Q}^* \mathbf{Y}$ .	Find ON matrices $\mathbf{Q}_c$ and $\mathbf{Q}_r$ such that $\mathbf{Y}_c = \mathbf{Q}_c \mathbf{Q}_c^* \mathbf{Y}_c$ and $\mathbf{Y}_r = \mathbf{Q}_r \mathbf{Q}_r^* \mathbf{Y}_r$ .
Solve for $\mathbf{C}$ the linear system $\mathbf{Q}^* \mathbf{Y} = \mathbf{C} (\mathbf{Q}^* \mathbf{G})$ .	Solve for $\mathbf{C}$ the linear systems $(\mathbf{G}_r^* \mathbf{Q}_c) \mathbf{C} = (\mathbf{Y}_r^* \mathbf{Q}_r)$ and $\mathbf{C} (\mathbf{Q}_r^* \mathbf{G}_c) = (\mathbf{Q}_c^* \mathbf{Y}_c)$
Factor $\mathbf{C}$ so that $\mathbf{C} = \hat{\mathbf{U}} \mathbf{D} \hat{\mathbf{U}}^*$ .	Factor $\mathbf{C}$ so that $\mathbf{C} = \hat{\mathbf{U}} \mathbf{D} \hat{\mathbf{V}}^*$ .
Form $\mathbf{U} = \mathbf{Q} \hat{\mathbf{U}}$ .	Form $\mathbf{U} = \mathbf{Q}_c \hat{\mathbf{U}}$ and $\mathbf{V} = \mathbf{Q}_r \hat{\mathbf{V}}$ .
<b>Output:</b> $\mathbf{A} \approx \mathbf{U} \mathbf{D} \mathbf{U}^*$	<b>Output:</b> $\mathbf{A} \approx \mathbf{U} \mathbf{D} \mathbf{V}^*$

**References:** Woolfe, Liberty, Rokhlin, and Tygert (2008), Clarkson and Woodruff (2009), Halko, Martinsson and Tropp (2009).



COST OF RANDOMIZED METHODS VS. CLASSICAL (DETERMINISTIC) METHODS:

**Case 1 —  $A$  is given as an array of numbers that fits in RAM (“small matrix”):**

Classical methods (e.g. Golub-Businger) have cost  $O(mnk)$ . The basic randomized method described also has  $O(mnk)$  cost, but with a much lower pre-factor since the matrix-matrix multiply is very fast. (The price we pay is some reduction in accuracy.)

**It gets better:** The cost can be reduced to  $O(mn \log k)$ ! The key is to replace the Gaussian random matrix by a “structured” random matrix. For instance,  $\mathbf{R}$  can be a sub-sampled randomized Fourier transform, which can be applied rapidly using variations of the FFT.

Example of a *subsampled random Fourier Transform (SRFT)*

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

- $\mathbf{D}$  is a diagonal matrix whose entries are i.i.d. random variables drawn from a uniform distribution on the unit circle in  $\mathbb{C}$ .
- $\mathbf{F}$  is the discrete Fourier transform,  $\mathbf{F}_{pq} = \frac{1}{\sqrt{n}} e^{-2\pi i(p-1)(q-1)/n}$ .
- $\mathbf{S}$  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  $\mathbf{D}\mathbf{F}$ .)

**References:** Ailon and Chazelle (2006); Liberty, Rokhlin, Tygert, and Woolfe (2006).

To achieve overall complexity  $O(mn \log(k))$ , we need to use the “single-pass” scheme so as to replace both matrix-matrix multiplications involving  $\mathbf{A}$  by “fast” operations.

*Input:* An  $m \times n$  matrix  $\mathbf{A}$ , and a rank  $k$ .

*Output:* Matrices  $\mathbf{U}$ ,  $\mathbf{V}$ ,  $\mathbf{D}$  that form a rank- $k$  SVD such that  $\mathbf{A} \approx \mathbf{U} \mathbf{D} \mathbf{V}^*$ .

1. Generate SRFT's  $\mathbf{R}_c$  and  $\mathbf{R}_r$  of sizes  $n \times k$ , and  $m \times k$ .
2. Form the sample matrices  $\mathbf{Y}_c = \mathbf{A} \mathbf{R}_c$  and  $\mathbf{Y}_r = \mathbf{A}^* \mathbf{R}_r$ .
3. Find ON matrices  $\mathbf{Q}_c$  and  $\mathbf{Q}_r$  such that  $\mathbf{Y}_c = \mathbf{Q}_c \mathbf{Q}_c^* \mathbf{Y}_c$  and  $\mathbf{Y}_r = \mathbf{Q}_r \mathbf{Q}_r^* \mathbf{Y}_r$ .
4. Solve for the  $k \times k$  matrix  $\mathbf{C}$  the systems

$$(\mathbf{Q}_c^* \mathbf{Y}_c) = \mathbf{C} (\mathbf{Q}_r^* \mathbf{R}_c) \quad \text{and} \quad (\mathbf{Q}_r^* \mathbf{Y}_r) = \mathbf{C}^* (\mathbf{Q}_c^* \mathbf{R}_r).$$

5. Compute the SVD of the small matrix  $\mathbf{C} = \hat{\mathbf{U}} \mathbf{D} \hat{\mathbf{V}}^*$  (and truncate if desired).
6. Form  $\mathbf{U} = \mathbf{Q}_c \hat{\mathbf{U}}$  and  $\mathbf{V} = \mathbf{Q}_r \hat{\mathbf{V}}$ .

**Observation 1:** Forming  $\mathbf{A} \mathbf{R}_c$  and  $\mathbf{A}^* \mathbf{R}_r$  in Step 2 has cost  $O(mn \log(k))$ .

**Observation 2:** All other steps cost at most  $O((m+n)k^2)$ .

**Note:** In Lecture 2, we will present a slightly different  $O(mn \log k)$  algorithm based on the *interpolative decomposition*.

## Practical speed of $O(mn \log k)$ complexity randomized SVD

Consider the task of computing a rank- $k$  SVD of a matrix  $\mathbf{A}$  of size  $n \times n$ .

$t^{(\text{direct})}$  Time for classical (Golub-Businger) method —  $O(k n^2)$

$t^{(\text{srft})}$  Time for randomized method with an SRFT —  $O(\log(k) n^2)$

$t^{(\text{gauss})}$  Time for randomized method with a Gaussian matrix —  $O(k n^2)$

$t^{(\text{svd})}$  Time for a full SVD —  $O(n^3)$

We will show the

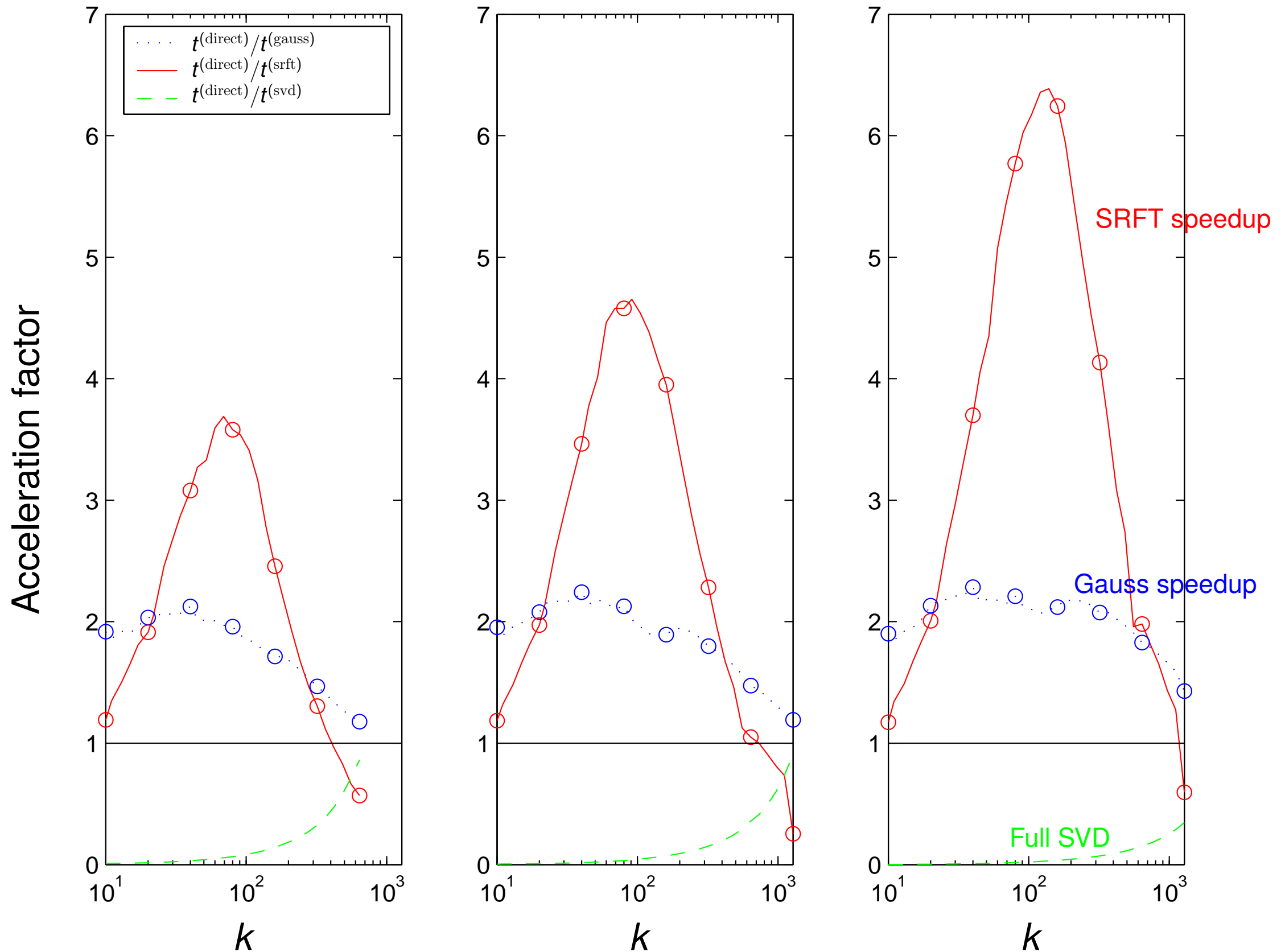
$$\text{acceleration factors: } \frac{t^{(\text{direct})}}{t^{(\text{srft})}} \quad \frac{t^{(\text{direct})}}{t^{(\text{gauss})}} \quad \frac{t^{(\text{direct})}}{t^{(\text{svd})}}$$

for different values of  $n$  and  $k$ .

$n = 1\,024$

$n = 2\,048$

$n = 4\,096$



**Observe:** Large speedups (up to a factor 6!) for moderate size matrices.

**Goal:** Given an  $m \times n$  matrix  $\mathbf{A}$ , compute an approximate rank- $k$  SVD  $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$ .

## Algorithm:

1. Draw an  $n \times k$  Gaussian random matrix  $\mathbf{R}$ . `R = randn(n,k)`
2. Form the  $m \times k$  sample matrix  $\mathbf{Y} = \mathbf{A}\mathbf{R}$ . `Y = A * R`
3. Form an  $m \times k$  orthonormal matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{Q}\mathbf{R}$ . `[Q, R] = qr(Y)`
4. Form the  $k \times n$  matrix  $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ . `B = Q' * A`
5. Compute the SVD of the small matrix  $\mathbf{B}$ :  $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$ . `[Uhat, Sigma, V] = svd(B,0)`
6. Form the matrix  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ . `U = Q * Uhat`

## Notes:

- The algorithm interacts with  $\mathbf{A}$  *only via the matrix-matrix multiplication*.
- This simple algorithm works well for many different types of problems.  $\mathbf{A}$  could be dense or sparse. It could be stored in RAM, on a hard drive, on a distributed memory parallel computer, on a GPU, etc.
- Today, we will discuss *costs* and *accuracy* of the method. We will also describe randomized techniques for “structure preserving” factorizations.
- On Thursday, we will discuss the mathematical *theory*, and describe other applications of randomized projections, and “Johnson-Lindenstrauss” techniques.

COST OF RANDOMIZED METHODS VS. CLASSICAL (DETERMINISTIC) METHODS:

***Case 1 —  $\mathbf{A}$  is given as an array of numbers that fits in RAM (“small matrix”):***

Classical methods (e.g. Golub-Businger) have cost  $O(mnk)$ . The basic randomized method described also has  $O(mnk)$  cost, but are very fast due to the efficiency of the matrix-matrix multiplication. However, the cost can be reduced to  $O(mn \log k)$  if a structured random matrix is used. For instance,  $\mathbf{R}$  can be a sub-sampled randomized Fourier transform, which can be applied rapidly using variations of the FFT.

***Case 2 —  $\mathbf{A}$  is given as an array of numbers on disk (“large matrix”):***

In this case, the relevant metric is memory access. Randomized methods access  $\mathbf{A}$  via sweeps over the entire matrix. With slight modifications, the randomized method can be executed in a *single pass* over the matrix. High accuracy can be attained with a small number of passes (say two, or five).

*(In contrast, classical (deterministic) methods require “random” access to matrix elements...)*

***Case 3 —  $\mathbf{A}$  and  $\mathbf{A}^*$  can be applied fast (“structured matrix”):***

Think of  $\mathbf{A}$  sparse, or sparse in the Fourier domain, or amenable to the Fast Multipole Method, etc. The classical competitor is in this case “Krylov methods”. Randomized methods tend to be more robust, and easier to implement in parallel environments. They are more easily blocked to reduce communication. However, Krylov methods sometimes lead to higher accuracy.

**Input:** An  $m \times n$  matrix  $\mathbf{A}$  and a target rank  $k$ .

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

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

(2) Form the  $m \times k$  **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  $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$ .

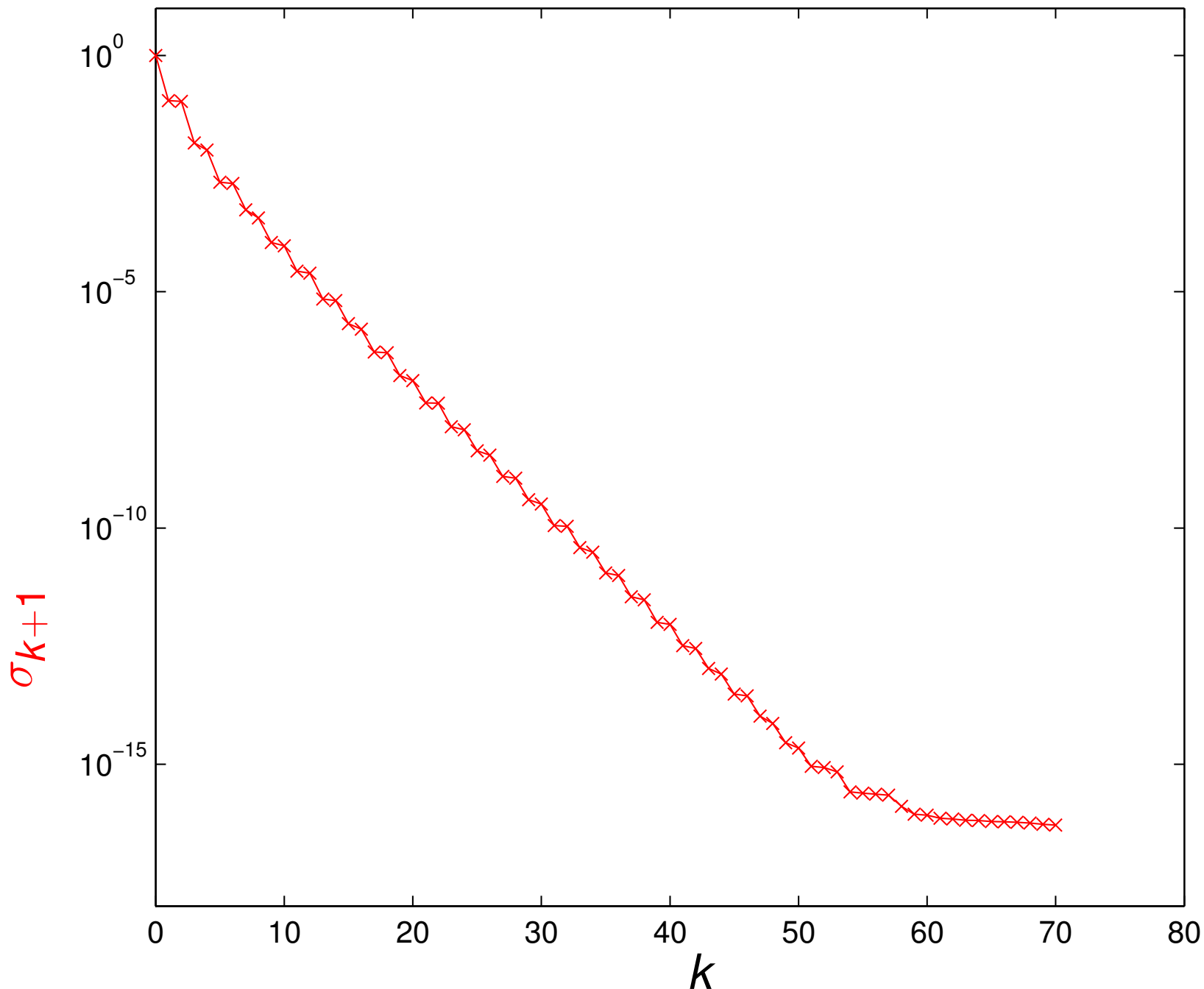
(6) Form  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ .

**Question:** What is the error  $e_k = \|\mathbf{A} - \mathbf{UDV}^*\|$ ? (Recall that  $e_k = \|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|$ .)

Let us first investigate some numerical examples. Then we will survey some rigorous mathematical results that provide bounds on the error and explain some of the phenomena we will see in the examples. (We outline the proofs in Lecture 3.)

## Example 1:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

---

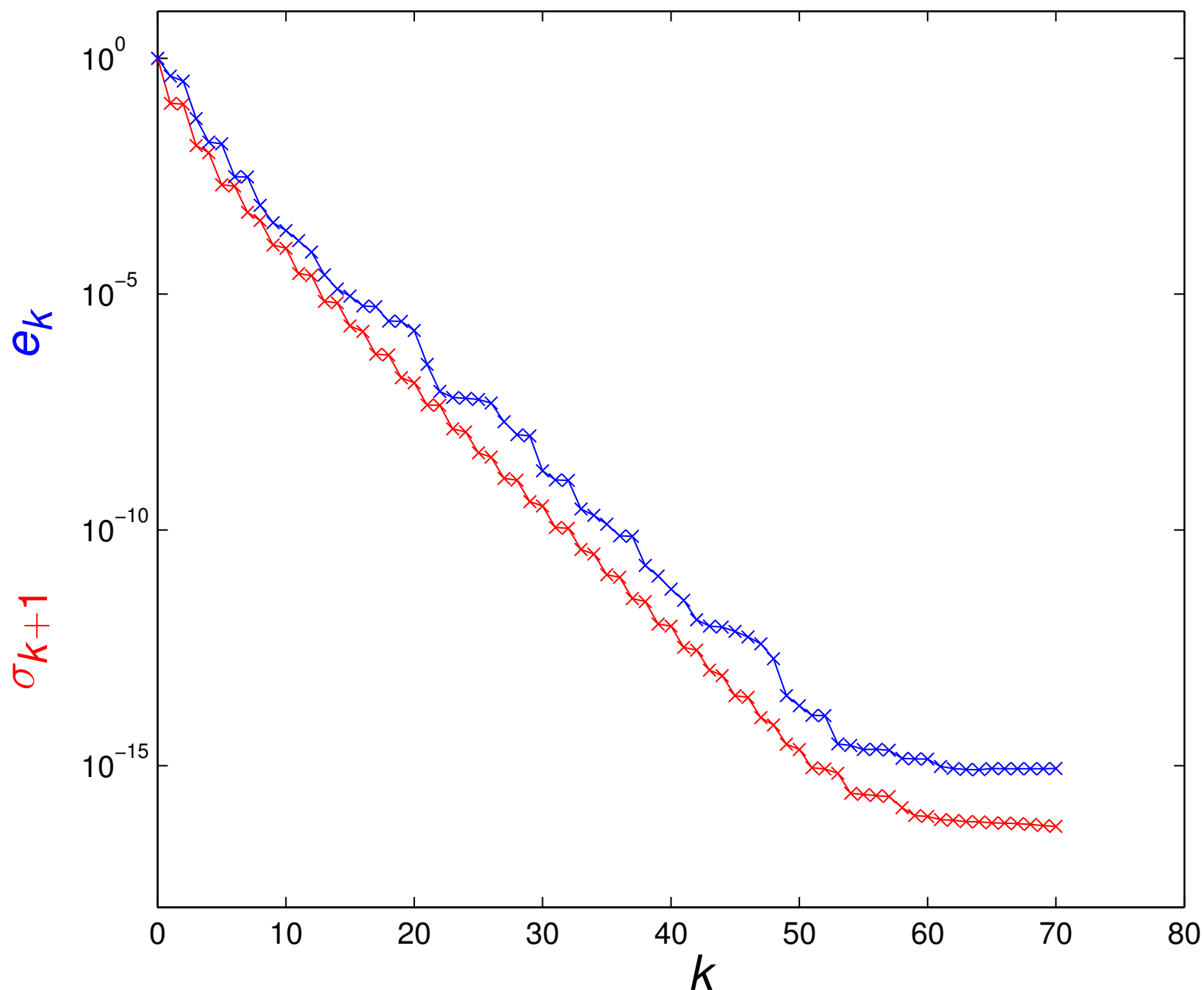
$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .



## Example 1:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

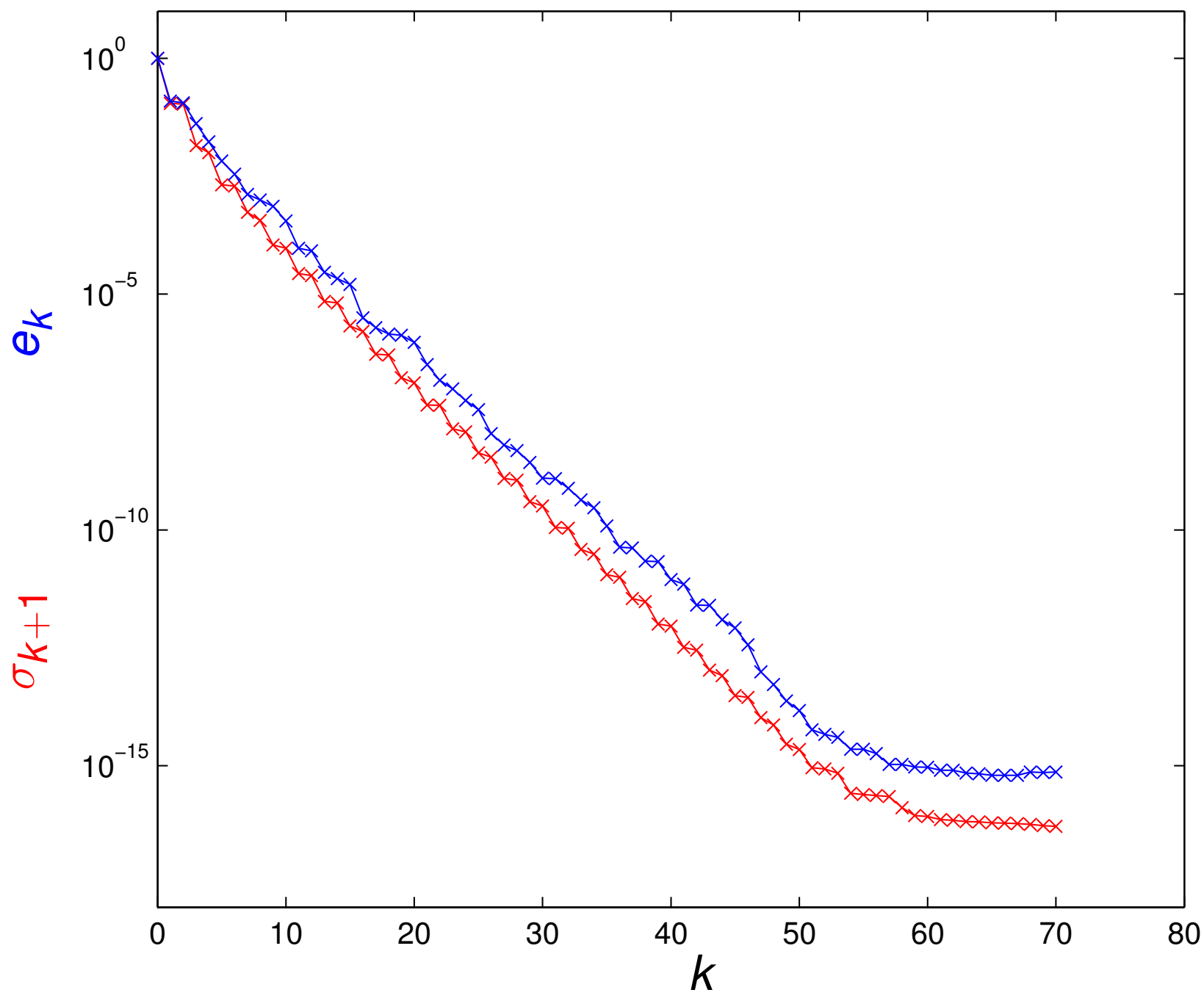
The blue line indicates the actual errors  $e_k$  incurred by one instantiation of the proposed method.

$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .

## Example 1:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

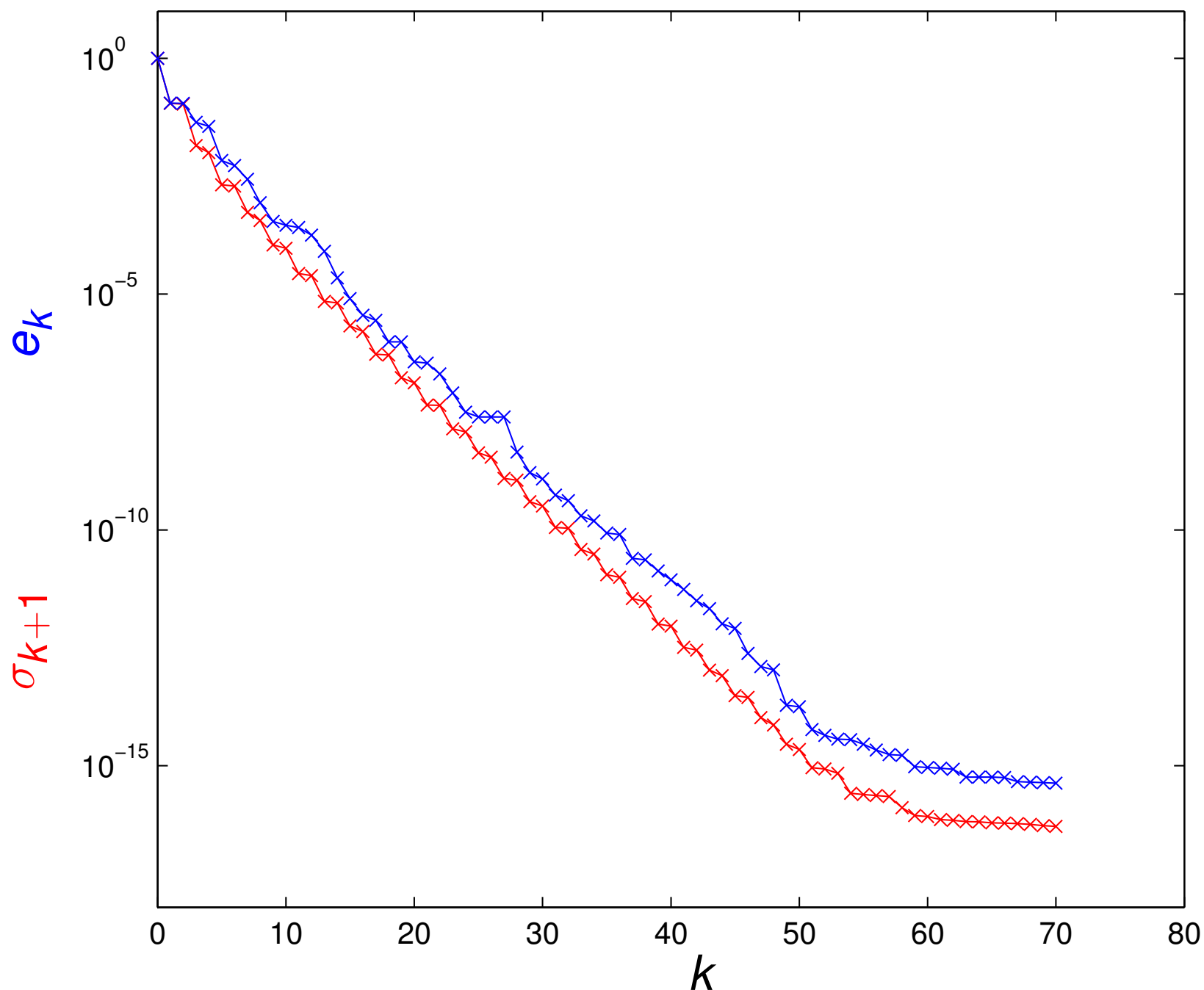
The blue line indicates the actual errors  $e_k$  incurred by a different instantiation of the proposed method.

$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .

## Example 1:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

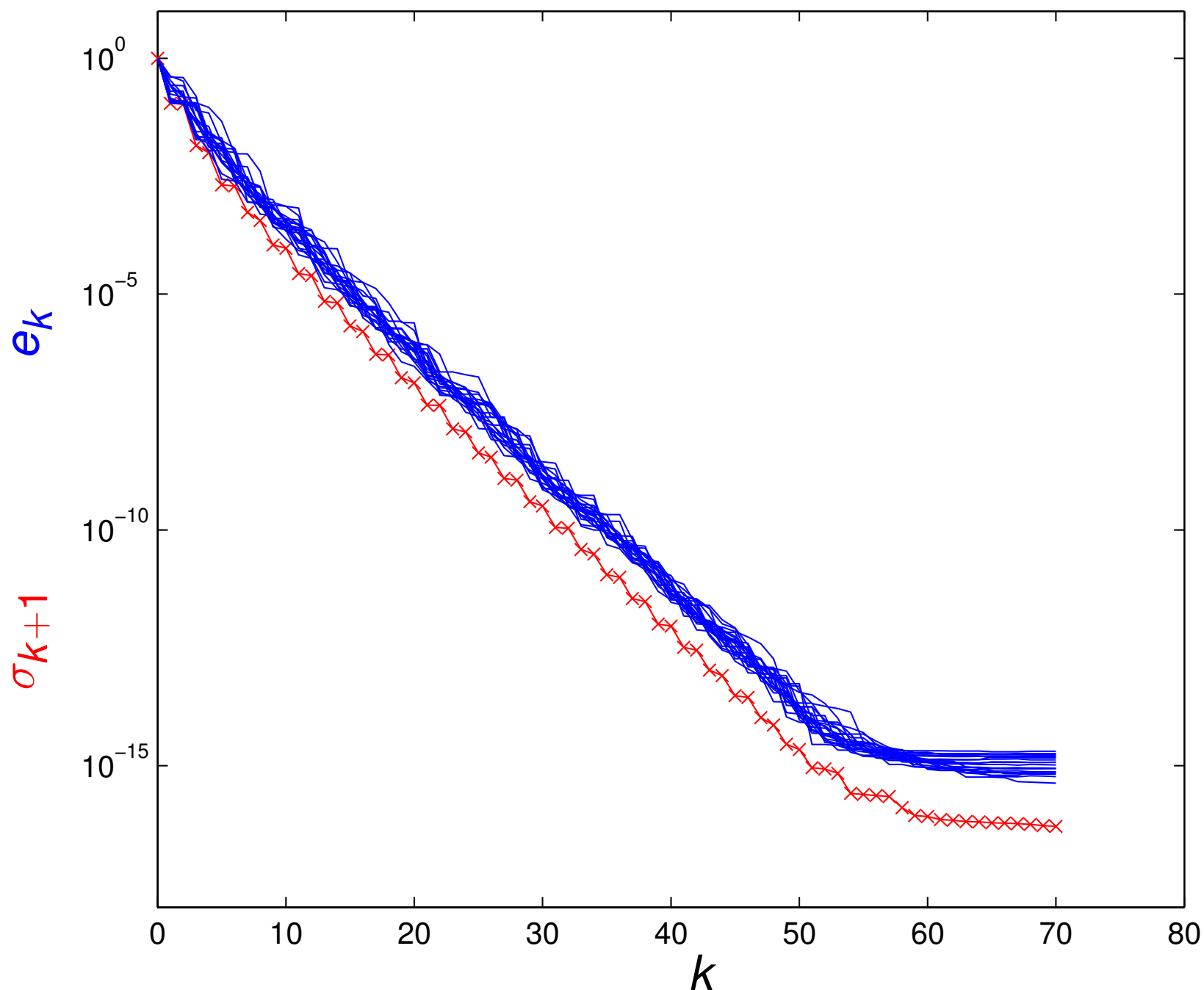
The blue line indicates the actual errors  $e_k$  incurred by a different instantiation of the proposed method.

$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .

## Example 1:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

The blue lines indicate the actual errors  $e_k$  incurred by 20 instantiations of the proposed method.

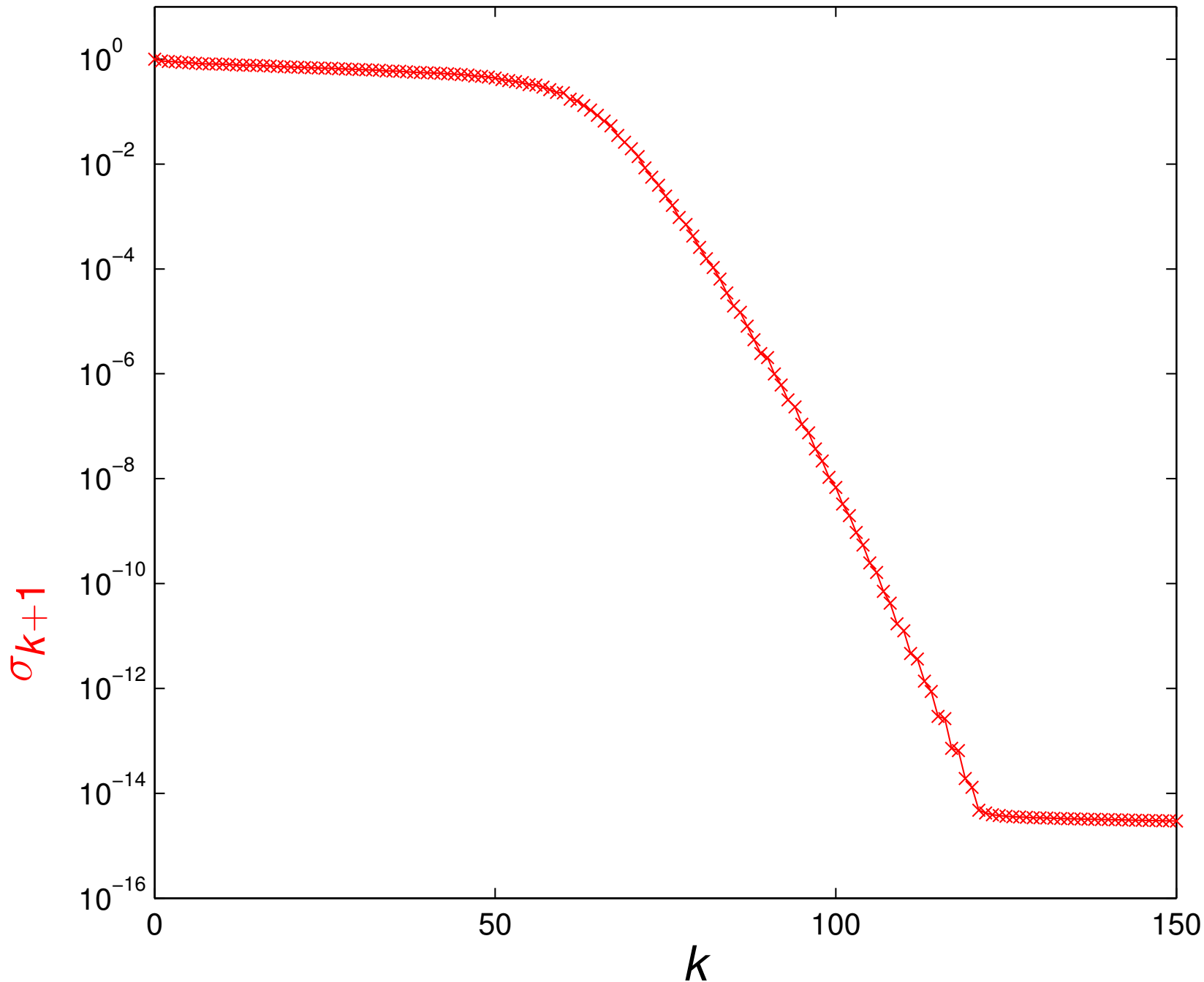
---

$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .

## Example 2:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

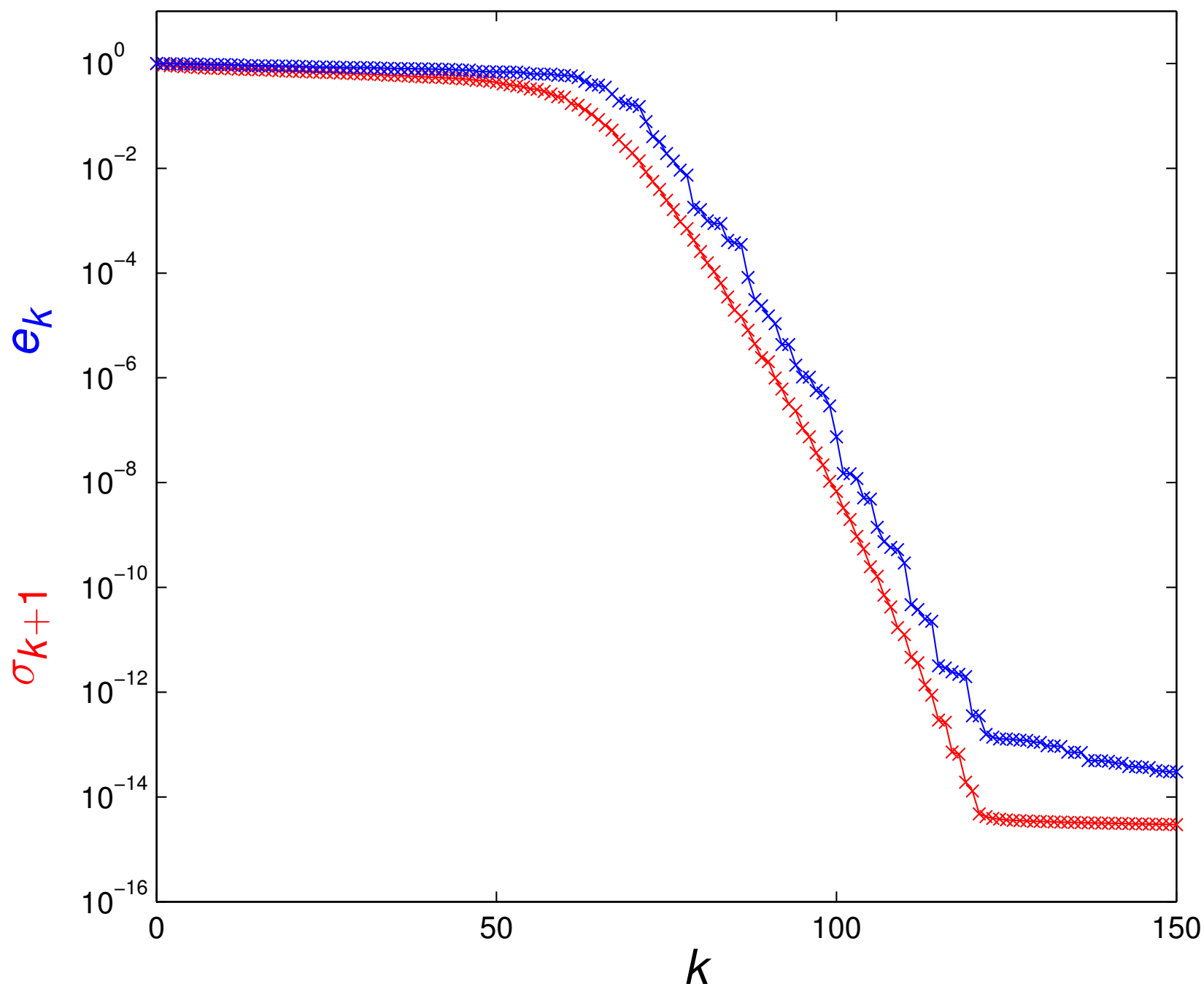
---

$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .

## Example 2:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

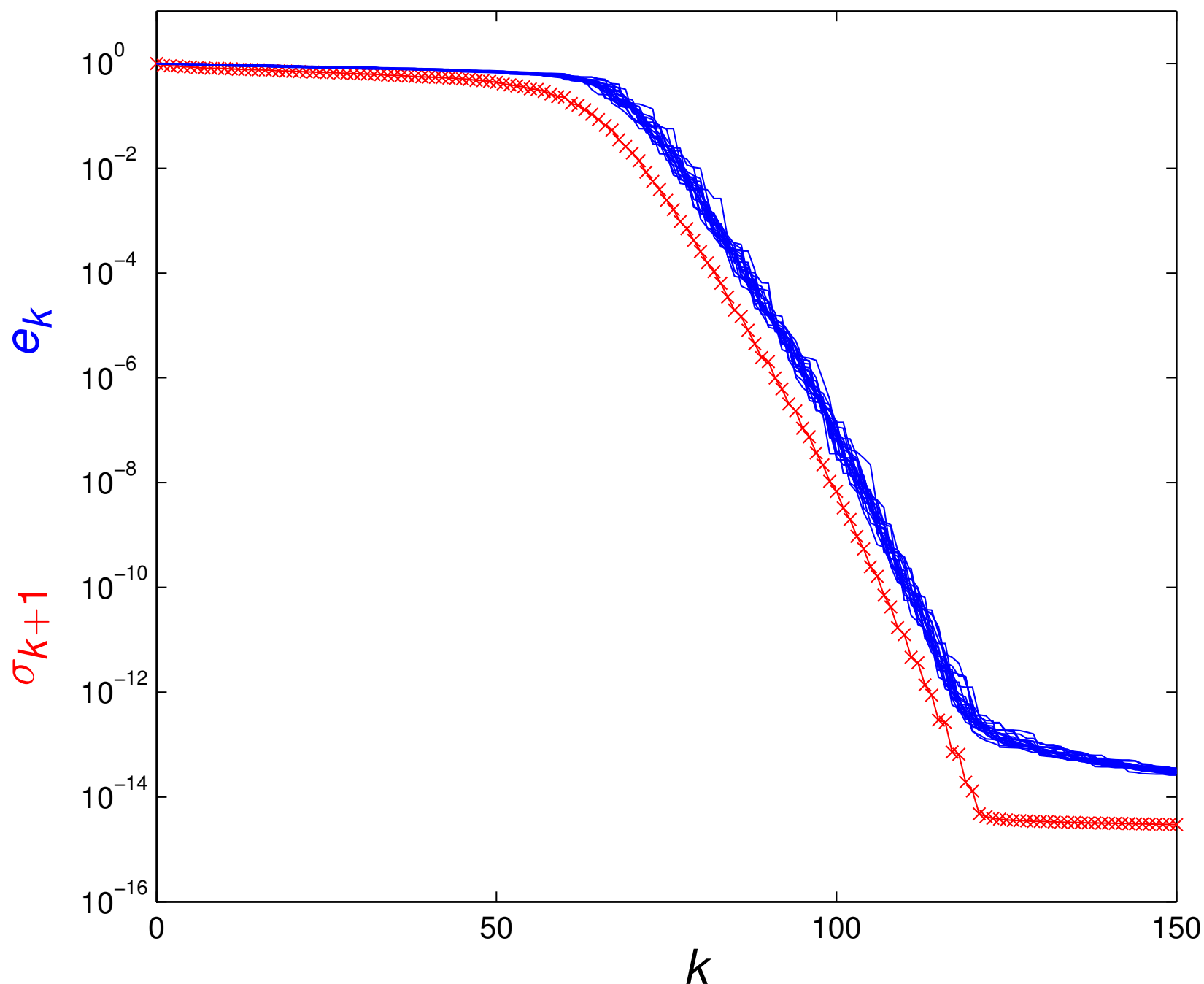
The blue line indicates the actual errors  $e_k$  incurred by one instantiation of the proposed method.

$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .

## Example 2:

We consider a  $1\,000 \times 1\,000$  matrix  $\mathbf{A}$  whose singular values are shown below:



The red line indicates the singular values  $\sigma_{k+1}$  of  $\mathbf{A}$ . These indicate the theoretically minimal approximation error.

The blue lines indicate the actual errors  $e_k$  incurred by 20 instantiations of the proposed method.

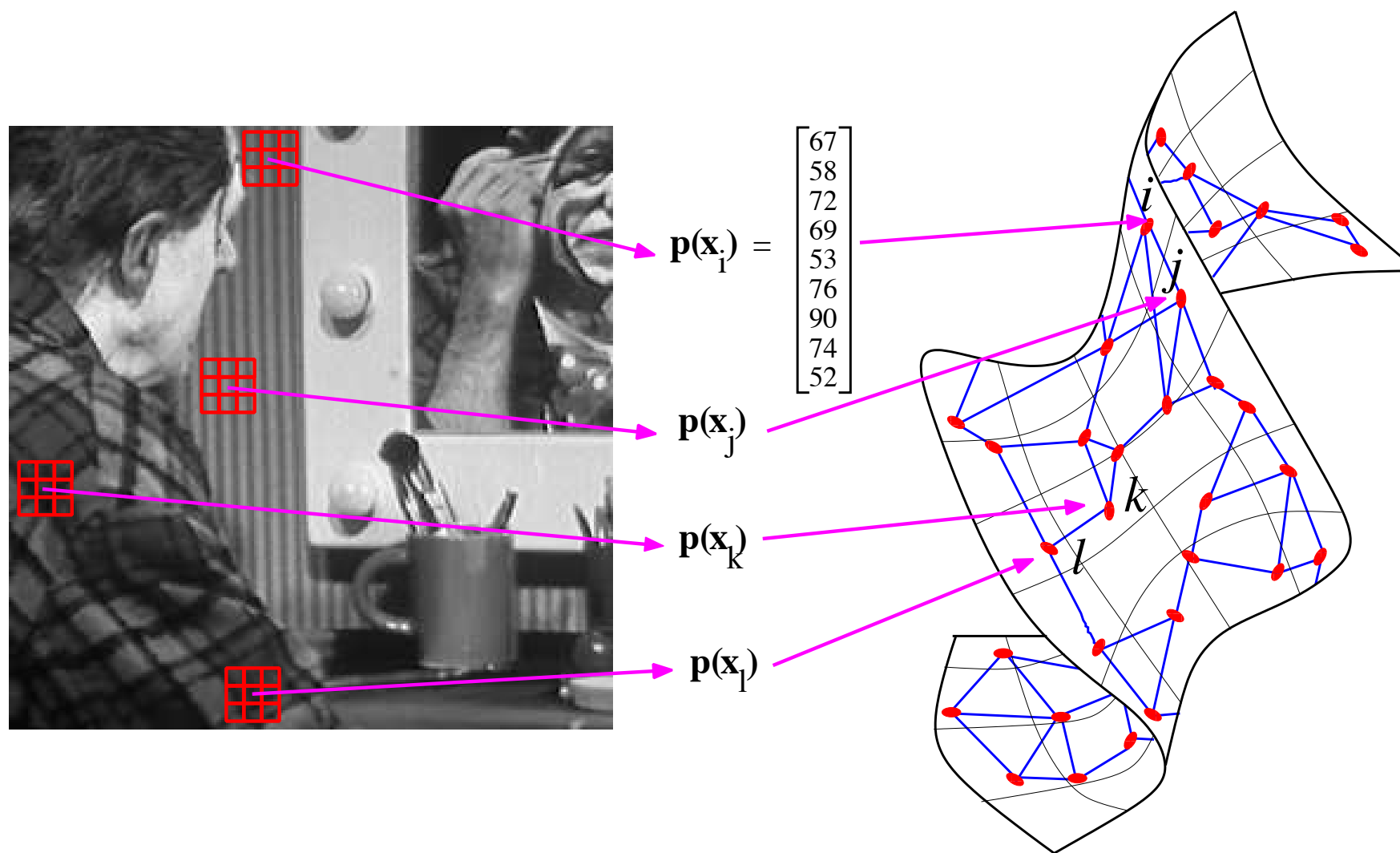
$\mathbf{A}$  is a discrete approximation of a certain compact integral operator normalized so that  $\|\mathbf{A}\| = 1$ .

Curiously, the nature of  $\mathbf{A}$  is in a strong sense irrelevant: the error distribution depends only on  $\{\sigma_j\}_{j=1}^{\min(m,n)}$ .

### Example 3:

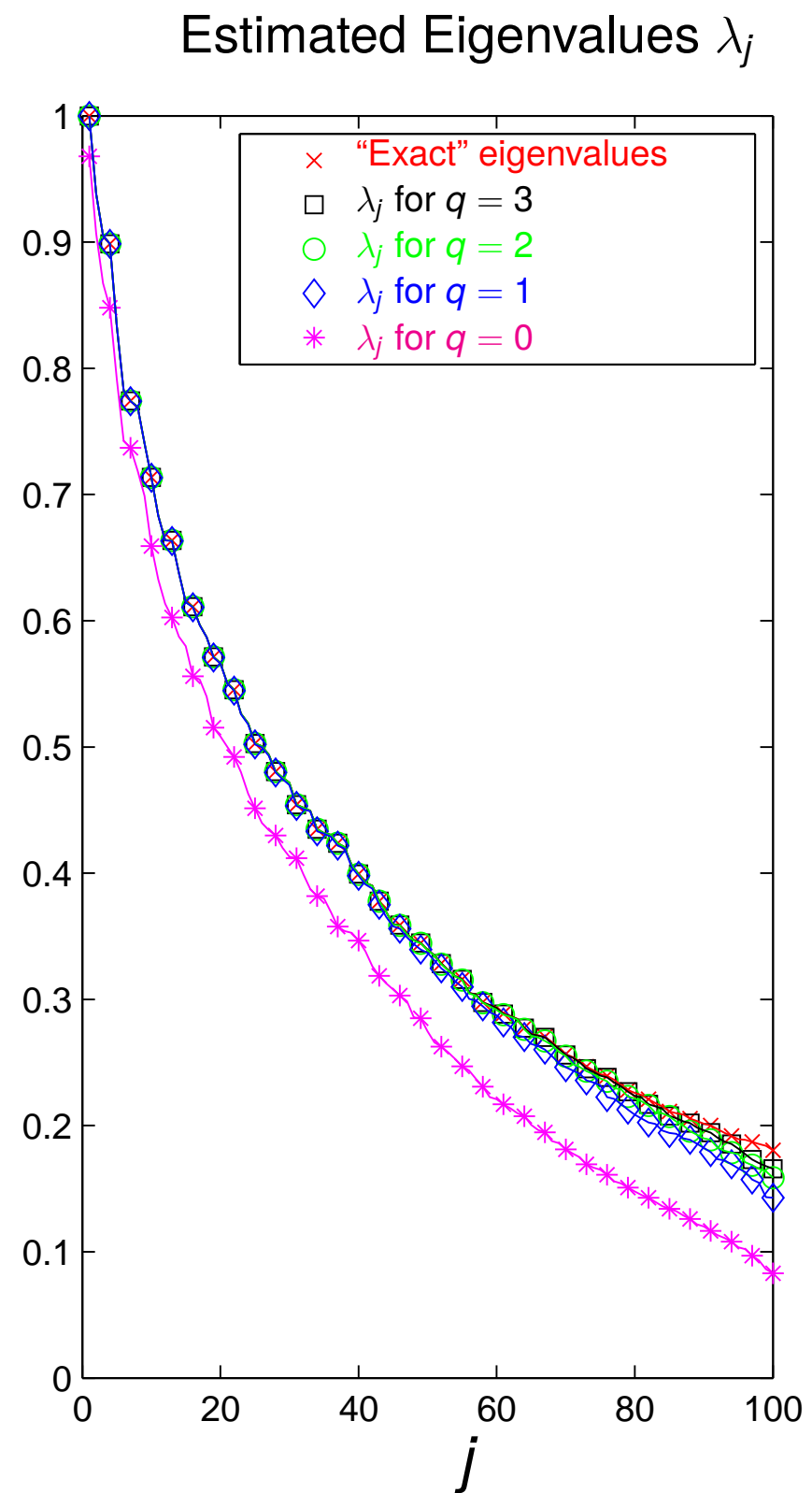
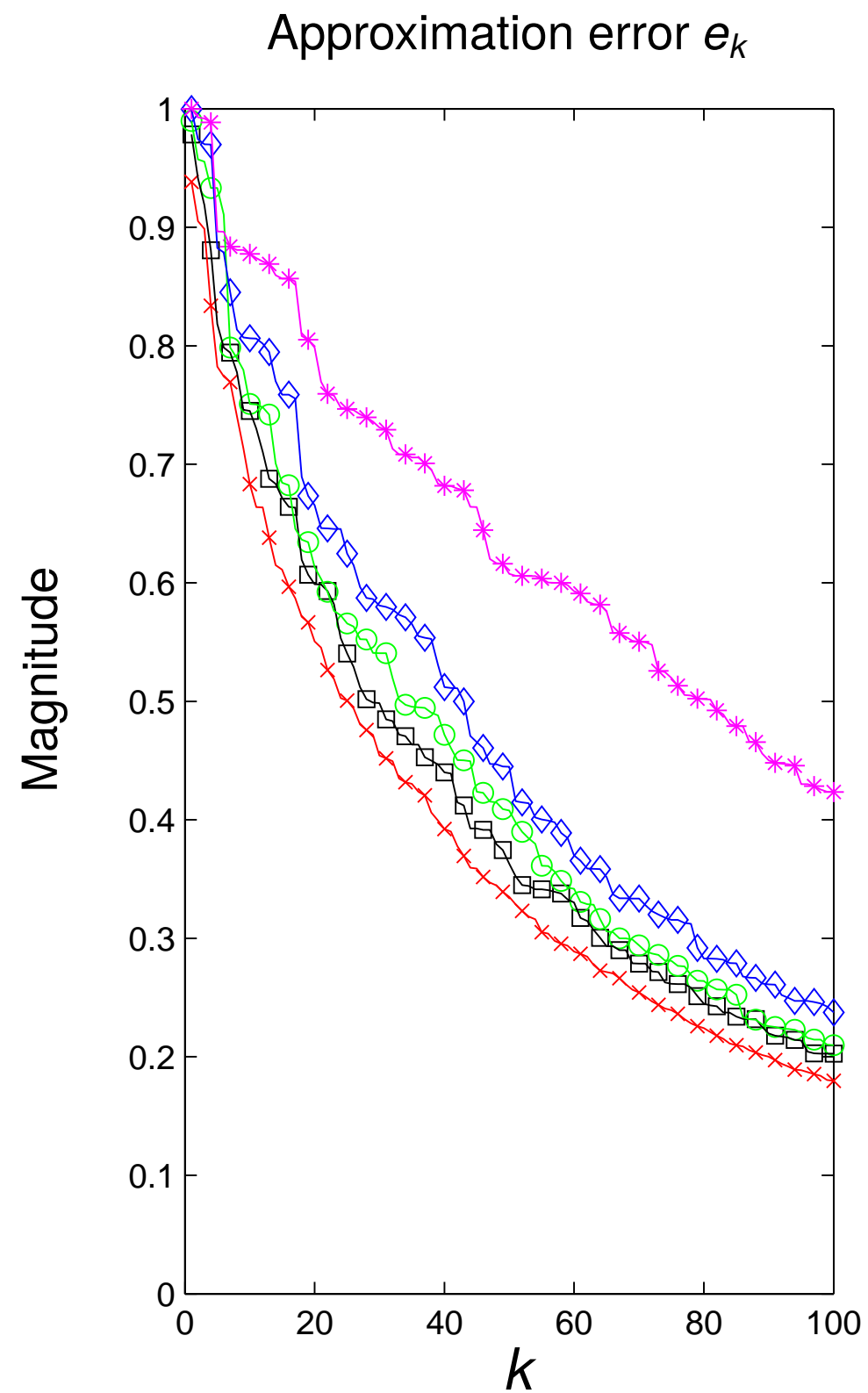
The matrix  $\mathbf{A}$  being analyzed is a  $9025 \times 9025$  matrix arising in a diffusion geometry approach to image processing.

To be precise,  $\mathbf{A}$  is a graph Laplacian on the manifold of  $3 \times 3$  patches.



*Joint work with François Meyer of the University of Colorado at Boulder.*





The pink lines illustrates the performance of the basic random sampling scheme. The errors are huge, and the estimated eigenvalues are much too small.

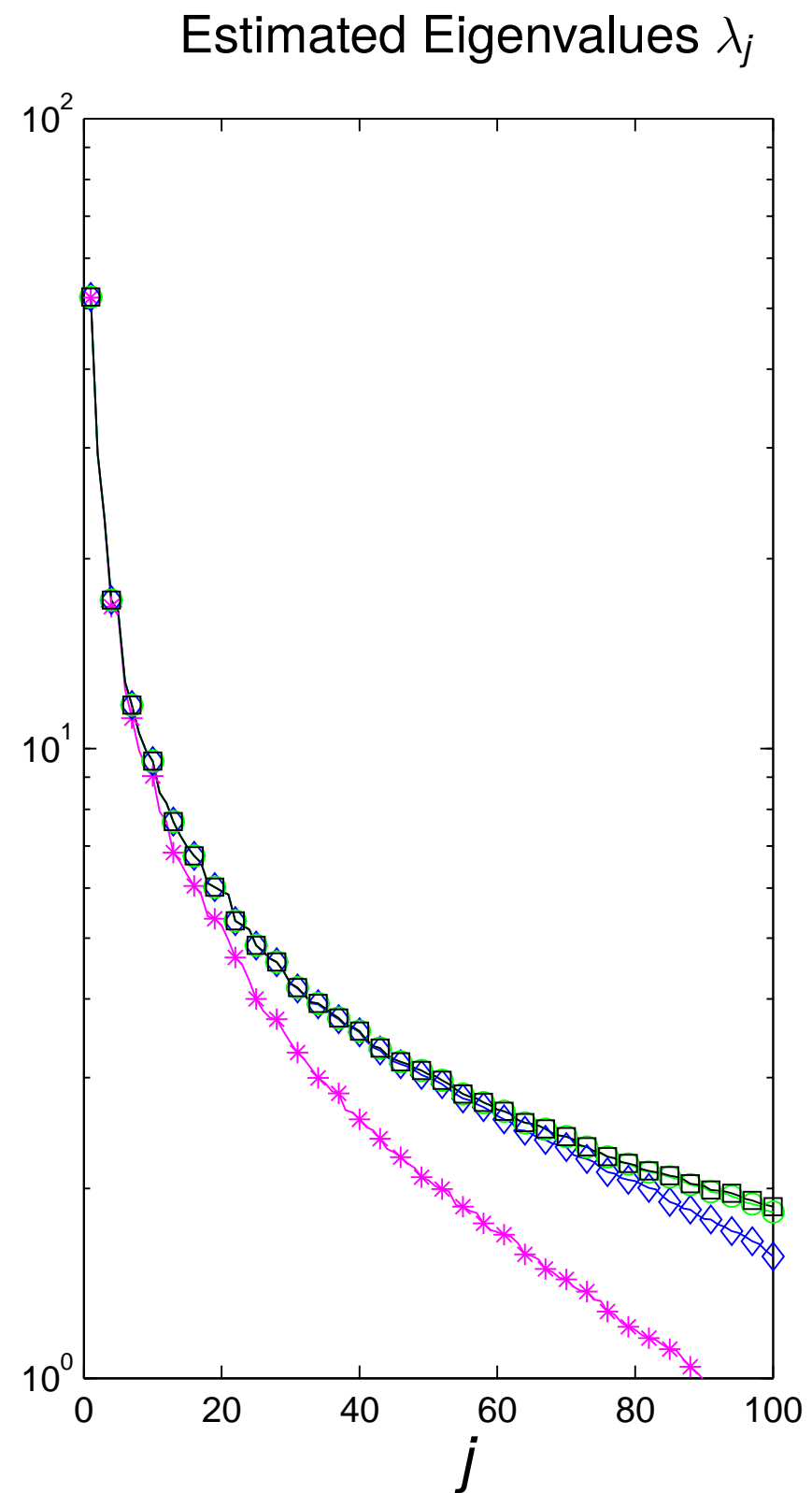
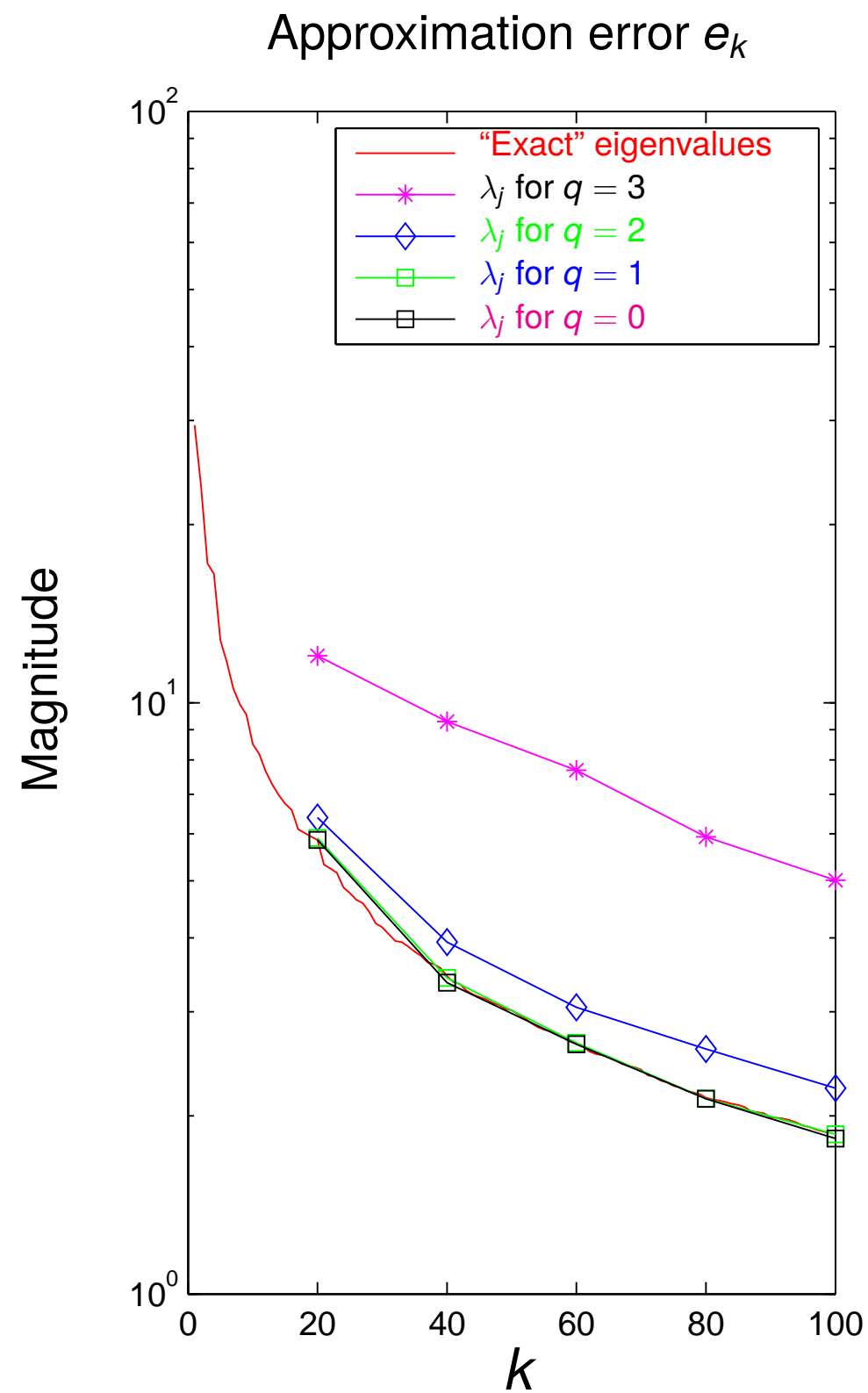
## Example 4: “Eigenfaces”

We next process a data base containing  $m = 7\,254$  pictures of faces

Each image consists of  $n = 384 \times 256 = 98\,304$  gray scale pixels.

We center and scale the pixels in each image, and let the resulting values form a column of a  $98\,304 \times 7\,254$  data matrix  $\mathbf{A}$ .

The left singular vectors of  $\mathbf{A}$  are the so called *eigenfaces* of the data base.



The pink lines illustrates the performance of the basic random sampling scheme. Again, the errors are huge, and the estimated eigenvalues are much too small.

**Input:** An  $m \times n$  matrix  $\mathbf{A}$  and a target rank  $k$ .

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

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

(2) Form the  $m \times k$  **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  $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$ .

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

**Question:** What is the error  $e_k = \|\mathbf{A} - \mathbf{UDV}^*\|$ ? (Recall that  $e_k = \|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|$ .)

**Eckart-Young theorem:**  $e_k$  is bounded from below by the singular value  $\sigma_{k+1}$  of  $\mathbf{A}$ .

**Question:** Is  $e_k$  close to  $\sigma_{k+1}$ ?

**Answer:** Lamentably, no. The expectation of  $\frac{e_k}{\sigma_{k+1}}$  is large, and has very large variance.

**Remedy:** Over-sample *slightly*. Compute  $k+p$  samples from the range of  $\mathbf{A}$ .

It turns out that  $p = 5$  or  $10$  is often sufficient.  $p = k$  is almost always more than enough.

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

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

## Bound on the expectation of the error for Gaussian test matrices

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  $\mathbf{R}$  denote an  $n \times (k + p)$  Gaussian matrix.

Let  $\mathbf{Q}$  denote the  $m \times (k + p)$  matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{R})$ .

If  $p \geq 2$ , then

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

and

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

*Ref: Halko, Martinsson, Tropp, 2009 & 2011*

### Observations:

- The error depends only on the singular values of  $\mathbf{A}$ .
- The error does not depend on the leading  $k$  singular values  $\{\sigma_j\}_{j=1}^k$  at all.

## Large deviation bound for the error for Gaussian test matrices

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  $\mathbf{R}$  denote an  $n \times (k + p)$  Gaussian matrix.

Let  $\mathbf{Q}$  denote the  $m \times (k + p)$  matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{R})$ .

If  $p \geq 4$ , and  $u$  and  $t$  are such that  $u \geq 1$  and  $t \geq 1$ , then

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \left(1 + t \sqrt{\frac{3k}{p+1}} + ut \frac{e \sqrt{k+p}}{p+1}\right) \sigma_{k+1} + \frac{te \sqrt{k+p}}{p+1} \left(\sum_{j>k} \sigma_j^2\right)^{1/2}$$

except with probability at most  $2t^{-p} + e^{-u^2/2}$ .

*Ref: Halko, Martinsson, Tropp, 2009 & 2011; Martinsson, Rokhlin, Tygert (2006)*

---

$u$  and  $t$  parameterize “bad” events — large  $u$ ,  $t$  is bad, but unlikely.

Certain choices of  $t$  and  $u$  lead to simpler results. For instance,

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \left(1 + 16\sqrt{1 + \frac{k}{p+1}}\right) \sigma_{k+1} + 8 \frac{\sqrt{k+p}}{p+1} \left(\sum_{j>k} \sigma_j^2\right)^{1/2},$$

except with probability at most  $3e^{-p}$ .

## Large deviation bound for the error for Gaussian test matrices

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  $\mathbf{R}$  denote an  $n \times (k + p)$  Gaussian matrix.

Let  $\mathbf{Q}$  denote the  $m \times (k + p)$  matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{R})$ .

If  $p \geq 4$ , and  $u$  and  $t$  are such that  $u \geq 1$  and  $t \geq 1$ , then

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \left(1 + t \sqrt{\frac{3k}{p+1}} + ut \frac{e \sqrt{k+p}}{p+1}\right) \sigma_{k+1} + \frac{te \sqrt{k+p}}{p+1} \left(\sum_{j>k} \sigma_j^2\right)^{1/2}$$

except with probability at most  $2t^{-p} + e^{-u^2/2}$ .

*Ref: Halko, Martinsson, Tropp, 2009 & 2011; Martinsson, Rokhlin, Tygert (2006)*

---

$u$  and  $t$  parameterize “bad” events — large  $u$ ,  $t$  is bad, but unlikely.

Certain choices of  $t$  and  $u$  lead to simpler results. For instance,

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \left(1 + 6\sqrt{(k+p) \cdot p \log p}\right) \sigma_{k+1} + 3\sqrt{k+p} \left(\sum_{j>k} \sigma_j^2\right)^{1/2},$$

except with probability at most  $3p^{-p}$ .

Let us look at the error bound a little closer:

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

*Case 1 — the singular values decay rapidly:* If  $(\sigma_j)$  decays sufficiently rapidly that  $\left(\sum_{j>k} \sigma_j^2\right)^{1/2} \approx \sigma_{k+1}$ , then we are fine — a minimal amount of over-sampling (say  $p = 5$  or  $p = k$ ) drives the error down close to the theoretically minimal value.

*Case 2 — the singular values do not decay rapidly:* In the worst case, we have

$$\left(\sum_{j>k} \sigma_j^2\right)^{1/2} \sim \sqrt{n-k} \sigma_{k+1}.$$

If  $n$  is large, and  $\sigma_{k+1}/\sigma_1$  is not that small, we could lose all accuracy.



*This is a common situation when you analyze noisy data. If there is, say, 5%*

*noise so that  $\sigma_j \approx 0.05 \|\mathbf{A}\|$  for  $j > k$ , then  $\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \approx 0.05 \|\mathbf{A}\| \sqrt{n}$ !*



## Power method for improving accuracy:

The error depends on how quickly the singular values decay.

The faster the singular values decay — the stronger the relative weight of the dominant modes in the samples.

**Idea:** The matrix  $(\mathbf{A} \mathbf{A}^*)^q \mathbf{A}$  has the same left singular vectors as  $\mathbf{A}$ , and its singular values are

$$\sigma_j((\mathbf{A} \mathbf{A}^*)^q \mathbf{A}) = (\sigma_j(\mathbf{A}))^{2q+1}.$$

Much faster decay — so let us use the sample matrix

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

instead of

$$\mathbf{Y} = \mathbf{A} \mathbf{G}.$$

**References:** Paper by Rokhlin, Szlam, Tygert (2008). Suggestions by Ming Gu. Also similar to “block power method,” and “block Lanczos.”

*Input:* An  $m \times n$  matrix  $\mathbf{A}$ , a target rank  $\ell$ , and a small integer  $q$ .

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

(1) Draw an  $n \times \ell$  **random matrix**  $\mathbf{R}$ .

(2) Form the  $m \times \ell$  **sample matrix**  $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{R}$ .

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

- Detailed (and, we believe, close to sharp) error bounds have been proven.

For instance, with  $\mathbf{A}^{\text{computed}} = \mathbf{UDV}^*$ , the expectation of the error satisfies:

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

*Reference: Halko, Martinsson, Tropp (2011).*

- The improved accuracy from the modified scheme comes at a cost;

$2q + 1$  passes over the matrix are required instead of 1.

However,  $q$  can often be chosen quite small in practice,  $q = 2$  or  $q = 3$ , say.

- The bound (4) assumes exact arithmetic.

To handle round-off errors, variations of subspace iterations can be used.

These are entirely numerically stable and achieve the same error bound.

## A numerically stable version of the “power method”:

*Input:* An  $m \times n$  matrix  $\mathbf{A}$ , a target rank  $\ell$ , and a small integer  $q$ .

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

Draw an  $n \times \ell$  Gaussian random matrix  $\mathbf{R}$ .

Set  $\mathbf{Q} = \text{orth}(\mathbf{AR})$

**for**  $i = 1, 2, \dots, q$

$\mathbf{W} = \text{orth}(\mathbf{A}^* \mathbf{Q})$

$\mathbf{Q} = \text{orth}(\mathbf{AW})$

**end for**

$\mathbf{B} = \mathbf{Q}^* \mathbf{A}$

$[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \text{svd}(\mathbf{B})$

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

**Note:** Algebraically, the method with orthogonalizations is identical to the “original” method where  $\mathbf{Q} = \text{orth}((\mathbf{AA}^*)^q \mathbf{AR})$ .

**Note:** This is a classic subspace iteration.

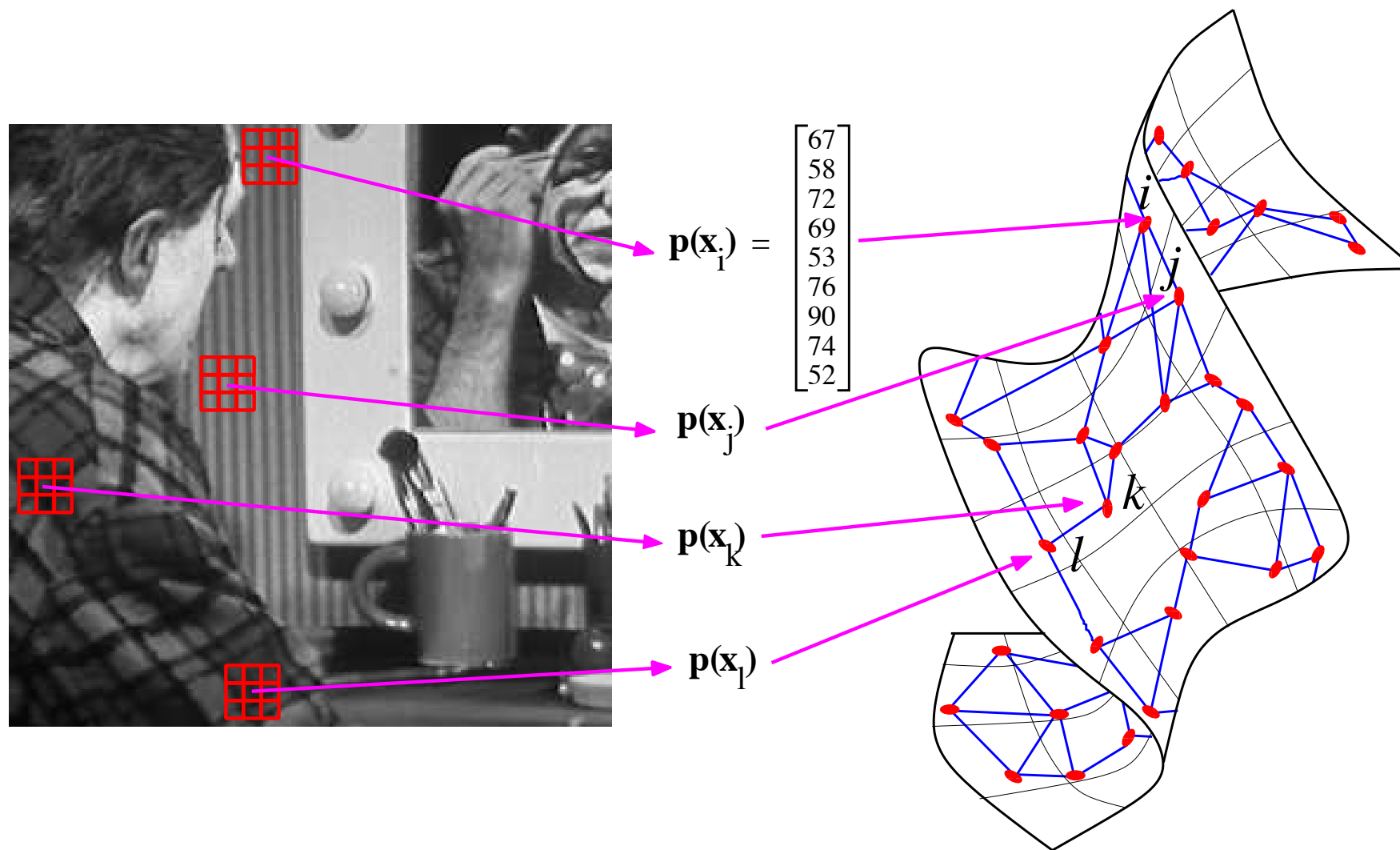
The novelty is the error analysis, and the finding that using a very small  $q$  is often fine.

(In fact, our analysis allows  $q$  to be zero...)

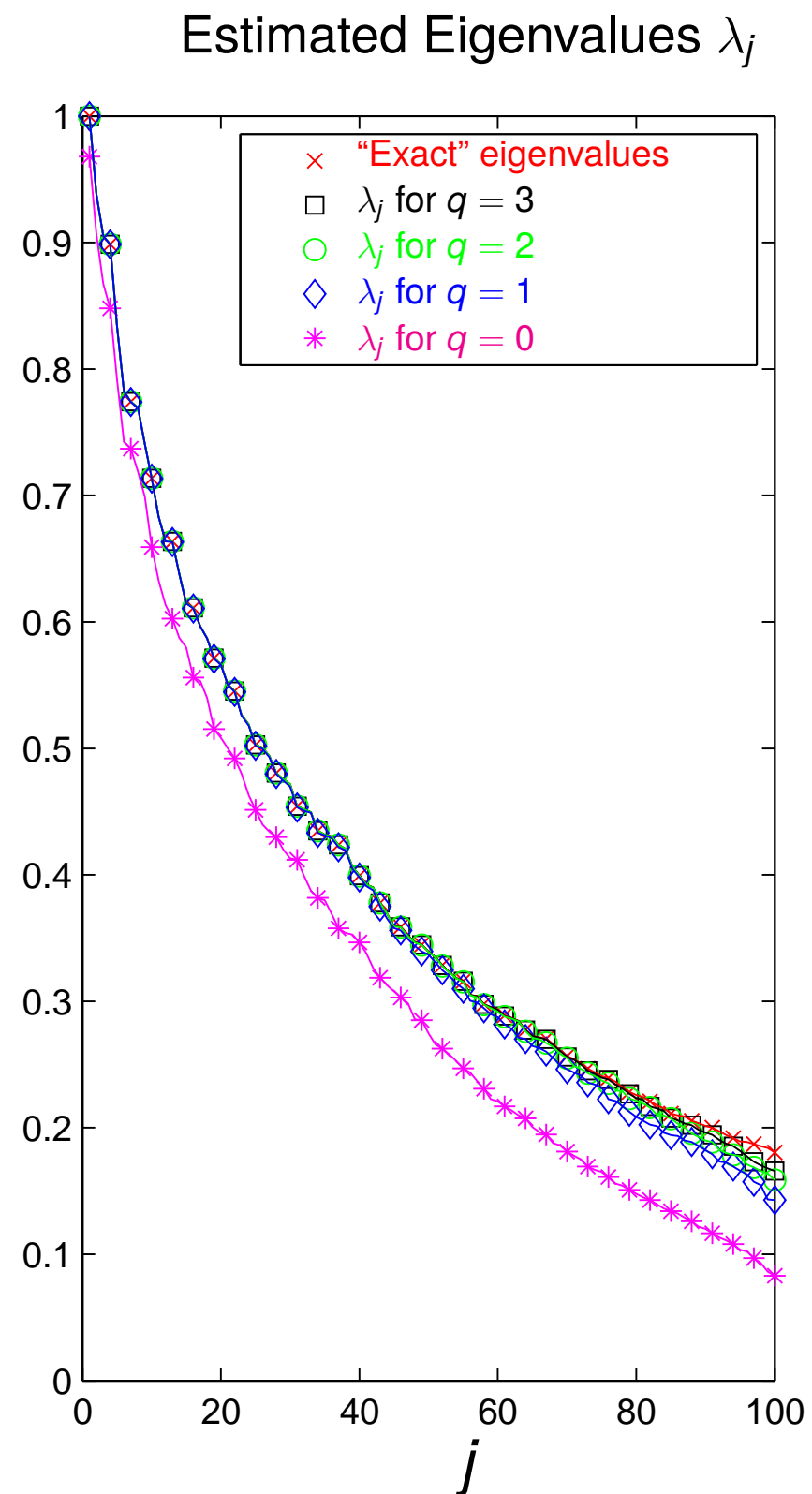
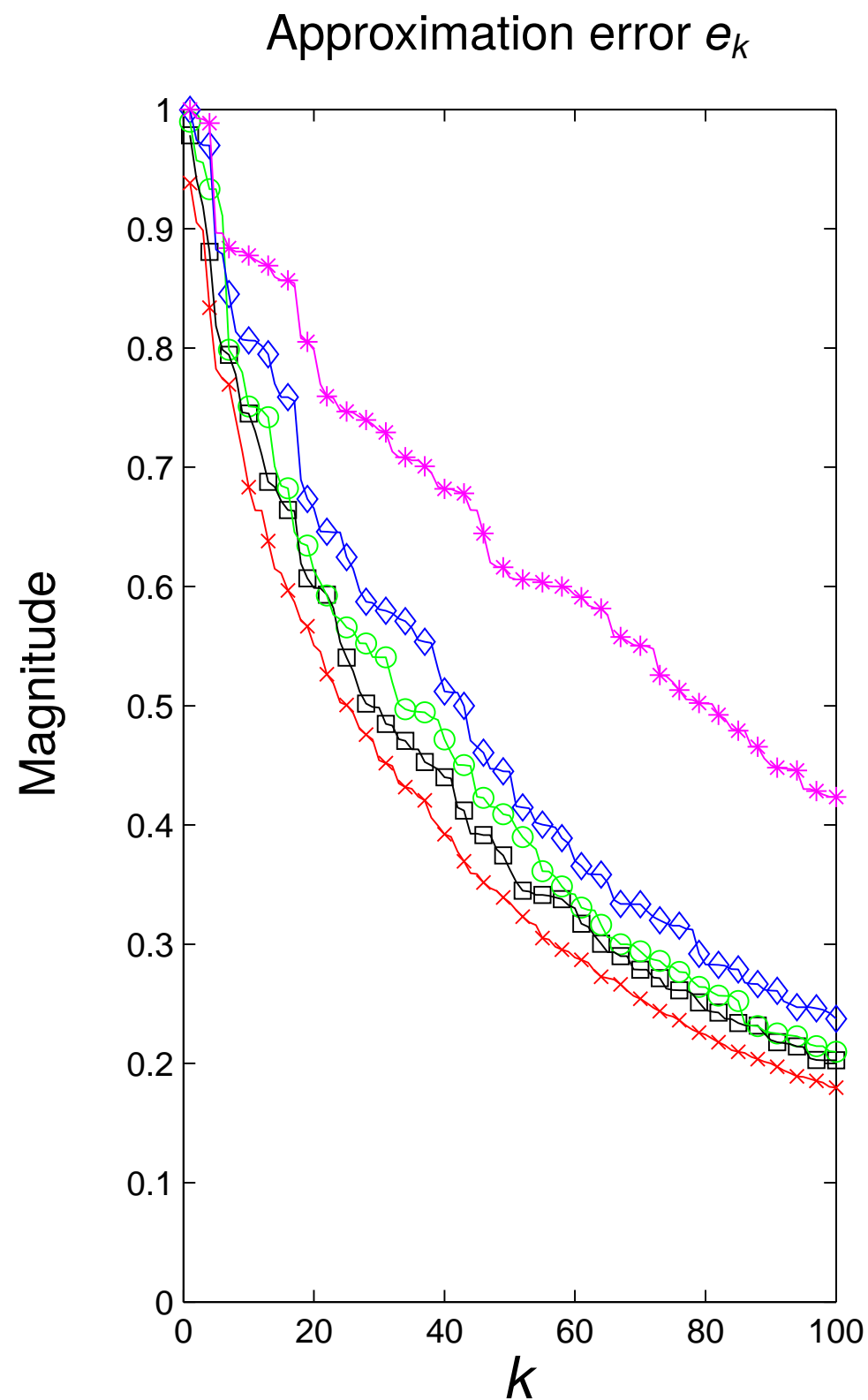
### Example 3 (revisited):

The matrix  $\mathbf{A}$  being analyzed is a  $9025 \times 9025$  matrix arising in a diffusion geometry approach to image processing.

To be precise,  $\mathbf{A}$  is a graph Laplacian on the manifold of  $3 \times 3$  patches.



*Joint work with François Meyer of the University of Colorado at Boulder.*



The pink lines illustrates the performance of the basic random sampling scheme. The errors for  $q = 0$  are huge, and the estimated eigenvalues are much too small. *But: The situation improves very rapidly as  $q$  is cranked up!*

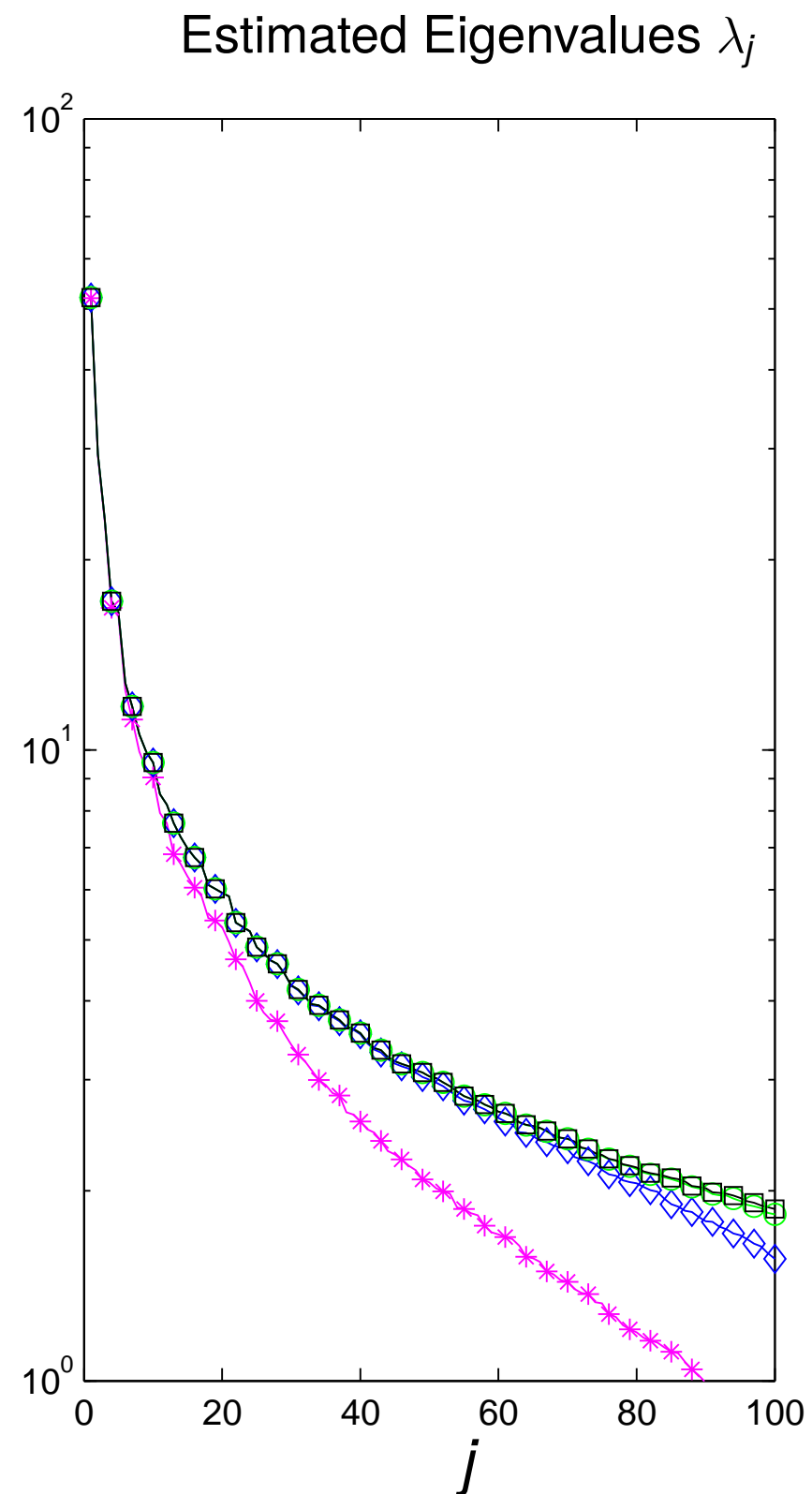
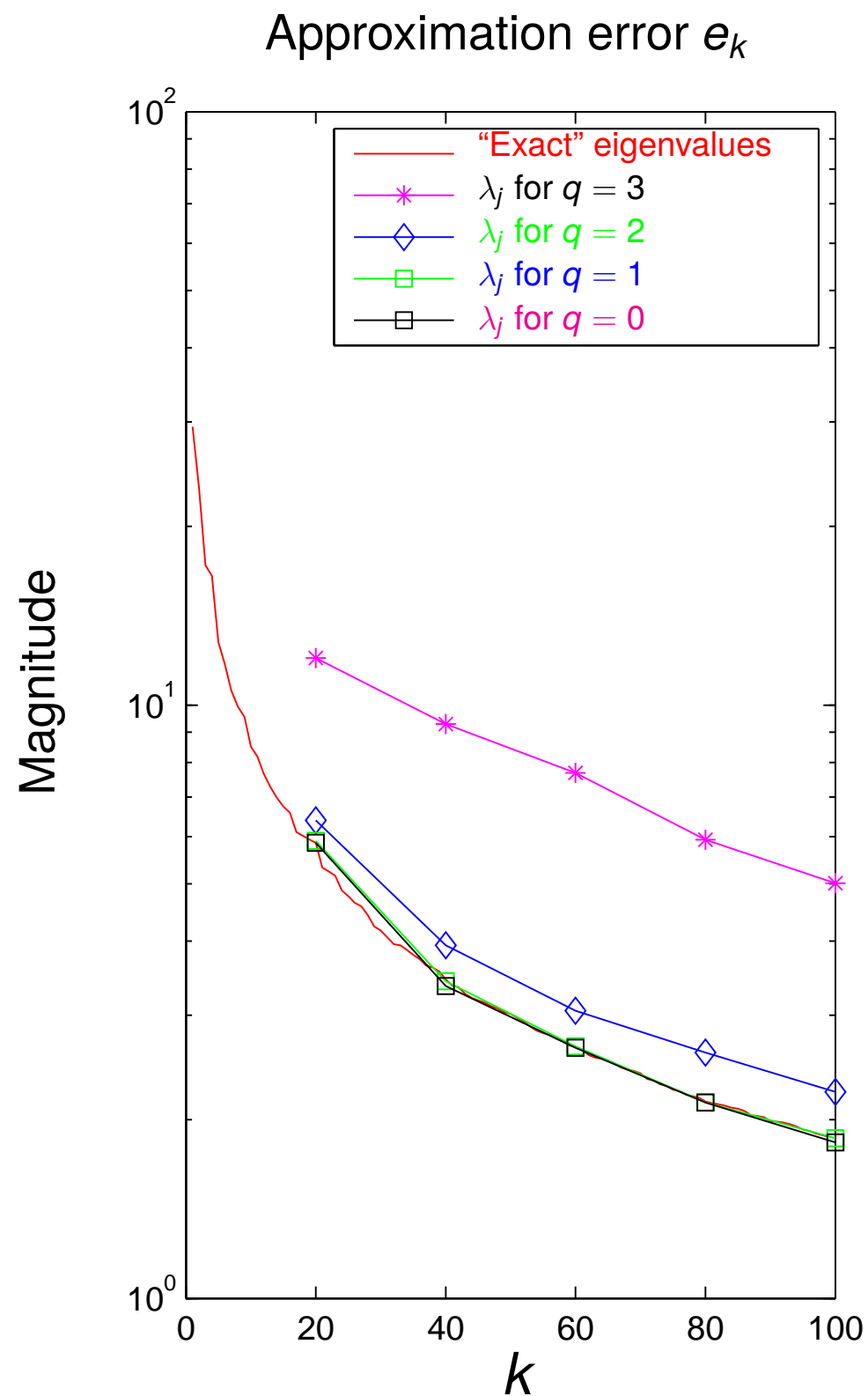
## Example 4 (revisited): “Eigenfaces”

We next process a data base containing  $m = 7\,254$  pictures of faces

Each image consists of  $n = 384 \times 256 = 98\,304$  gray scale pixels.

We center and scale the pixels in each image, and let the resulting values form a column of a  $98\,304 \times 7\,254$  data matrix  $\mathbf{A}$ .

The left singular vectors of  $\mathbf{A}$  are the so called *eigenfaces* of the data base.



The pink lines illustrates the performance of the basic random sampling scheme. Again, the errors are huge, and the estimated eigenvalues are much too small.  
*But: The situation improves very rapidly as  $q$  is cranked up!*

## Interpolative and CUR decompositions: The basic column ID

[This section has strong connections to the lectures of Petros Drineas.]

Any matrix  $\mathbf{A}$  of size  $m \times n$  and rank  $k$ , where  $k < \min(m, n)$ , admits a so called “interpolative decomposition (ID)” which takes the form

$$(5) \quad \begin{array}{ccccc} \mathbf{A} & = & \mathbf{C} & \mathbf{Z}, \\ m \times n & & m \times k & k \times n \end{array}$$

where the matrix  $\mathbf{C}$  is given by a subset of the columns of  $\mathbf{A}$  and where  $\mathbf{Z}$  is well-conditioned in a sense that we will make precise shortly. The ID has several advantages, as compared to, e.g., the QR or SVD factorizations:

- If  $\mathbf{A}$  is *sparse* or *non-negative*, then  $\mathbf{C}$  shares these properties.
- The ID requires *less memory* to store than either the QR or the SVD.
- Finding the indices associated with the spanning columns is often helpful in *data interpretation*.

One shortcoming of the ID is that when  $\mathbf{A}$  is not of precisely rank  $k$ , then the approximation error by the best possible rank- $k$  ID can be substantially larger than the theoretically minimal error. (In fact, the ID and the column pivoted QR factorizations are closely related, and they attain *exactly* the same minimal error.)



## Interpolative and CUR decompositions: Three flavors of IDs

Recall the *column-ID*:

$$(6) \quad \begin{array}{ccccc} \mathbf{A} & = & \mathbf{C} & \mathbf{Z}, \\ m \times n & & m \times k & k \times n \end{array}$$

where  $\mathbf{C} = \mathbf{A}(:, J_S)$  holds a subset of  $k$  columns of  $\mathbf{A}$ .

“Use  $k$  columns to span the column space of  $\mathbf{A}$ .”

---

There is an analogous *row-ID*:

$$(7) \quad \begin{array}{ccccc} \mathbf{A} & = & \mathbf{X} & \mathbf{R}, \\ m \times n & & m \times k & k \times n \end{array}$$

where  $\mathbf{R} = \mathbf{A}(I_S, :)$  holds a subset of  $k$  rows of  $\mathbf{A}$ .

“Use  $k$  rows to span the row space of  $\mathbf{A}$ .”

---

Finally, there is a *double-sided ID*:

$$(8) \quad \begin{array}{ccccc} \mathbf{A} & = & \mathbf{X} & \mathbf{A}_{\text{skel}} & \mathbf{Z}, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where  $\mathbf{A}_{\text{skel}} = \mathbf{A}(I_S, J_S)$  is a  $k \times k$  submatrix of  $\mathbf{A}$ .

“Use  $k$  columns to span the column space *and*  $k$  rows to span the row space.”

## Interpolative and CUR decompositions: Connection between ID and CUR

Recall from the previous slide the double-sided ID:

$$(9) \quad \mathbf{A} = \mathbf{X} \mathbf{A}_{\text{skel}} \mathbf{Z},$$

$m \times n \quad m \times k \quad k \times k \quad k \times n$

where  $\mathbf{A}_{\text{skel}} = \mathbf{A}(I_s, J_s)$  is a  $k \times k$  submatrix of  $\mathbf{A}$ . Also recall from the lectures by Petros Drineas the CUR decomposition:

$$(10) \quad \mathbf{A} = \mathbf{C} \mathbf{U} \mathbf{R},$$

$m \times n \quad m \times k \quad k \times k \quad k \times n$

where  $\mathbf{C} = \mathbf{A}(:, J_s)$  and  $\mathbf{R} = \mathbf{A}(I_s, :)$ .

Both (9) and (10) use the rows identified by  $I_s$  as a basis for the row-space, and the columns identified by  $J_s$  as a basis for the column space. Both of them share advantages in terms of preserving properties of  $\mathbf{A}$  (sparsity, non-negativity, etc) and in enabling data interpretation. For sparse matrices, the CUR is more storage efficient. (Although with a slight twist, the storage requirements of the ID can match those of CUR.) In situations where  $\mathbf{A}(I_s, J_s)$  is well-conditioned, either factorization can be used without difficulty. However, when  $\mathbf{A}(I_s, J_s)$  is ill-conditioned, the double-sided ID is a stable decomposition that is more immune to the effects of round-off errors and other numerical problems. Recall that common formulas for the “link matrix”  $\mathbf{U}$  include:

$$\mathbf{U} = \mathbf{A}(I_s, J_s)^{-1}, \quad \text{or} \quad \mathbf{U} = \mathbf{C}^\dagger \mathbf{A} \mathbf{R}^\dagger.$$

## Interpolative and CUR decompositions: Connection to column pivoted QR

Let  $\mathbf{A}$  be an  $m \times n$  matrix whose columns  $\{\mathbf{a}_j\}_{j=1}^n \in \mathbb{R}^m$  approximately lie in a space of dimension  $k$ .

**Question:** How do you find an index set  $J_S$  s.t.  $\mathbf{A}(:, J_S)$  forms a good basis for  $\text{Col}(\mathbf{A})$ ?

**Optimal answer:**  $J_S$  is the index set that maximizes the volume spanned by the vectors  $\mathbf{A}(:, J_S)$ . Very expensive to find!

**Leverage scores:** Randomized sampling using carefully chosen sampling weights provides an elegant solution with strong theoretical results in support. See lectures by Petros Drineas.

**Column pivoted QR (CPQR):** The problem is analogous to finding a basis for  $\text{span}\{\mathbf{a}_j\}_{j=1}^n$ . The classical text book solution is to apply Gram-Schmidt. This works quite well in most circumstances. It is communication intensive, however, so not well suited for very large matrices. The theory on approximation errors is complex, and in certain ways unsatisfactory.

**Randomized projection followed by CPQR:** Works very well in practice, slightly weaker theory. Well suited for very large matrices.

## Deriving a column ID from a partial column pivoted QR factorization

Recall that the Gram-Schmidt process can be described conveniently via the QR factorization. To be precise, after  $k$  steps of the Gram-Schmidt process, we have determined a factorization

$$(11) \quad \mathbf{A}(:, J) = \mathbf{Q}_1 \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{B} \end{bmatrix},$$

$$m \times n \quad m \times k \quad k \times k \quad k \times (n - k) \quad m \times k \quad m \times (n - k)$$

where  $J$  is a permutation of the indices  $[1, 2, \dots, n]$ , where  $\mathbf{R}_{11}$  is upper triangular, and where  $\mathbf{B}$  is a remainder matrix that is small in magnitude. Let us partition the index vector

$$J = [J_s \quad J_{\text{rem}}],$$

so that  $J_s$  is a vector identifying the  $k$  chosen “pivot columns.” Then by construction

$$\mathbf{A}(:, J_s) = \mathbf{Q}_1 \mathbf{R}_{11}.$$

Now let  $\mathbf{P}$  be the permutation matrix for which  $\mathbf{A}(:, J) = \mathbf{A}\mathbf{P}$ . Then (11) can be written

$$\mathbf{A}\mathbf{P} = \mathbf{Q}_1 \mathbf{R}_{11} \begin{bmatrix} \mathbf{I} & \mathbf{R}_{11}^{-1} \mathbf{R}_{12} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{B} \end{bmatrix}.$$

Right multiplying by  $\mathbf{P}^*$  (recall that  $\mathbf{P}\mathbf{P}^* = \mathbf{I}$  since  $\mathbf{P}$  is unitary) we get

$$\mathbf{A} = \underbrace{\mathbf{A}(:, J_s)}_{=: \mathbf{C}} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{R}_{11}^{-1} \mathbf{R}_{12} \end{bmatrix} \mathbf{P}^*}_{=: \mathbf{Z}} + \begin{bmatrix} \mathbf{0} & \mathbf{B} \end{bmatrix} \mathbf{P}^*.$$

## Computing a double-sided ID, or a CUR decomposition

Given an  $m \times n$  matrix  $\mathbf{A}$ , we seek to compute a double-sided ID or CUR decomposition.

**Step 1:** Compute a column ID of  $\mathbf{A}$  via column pivoted Gram-Schmidt:

$$(12) \quad \begin{array}{ccccccc} \mathbf{A} & = & \mathbf{C} & \mathbf{Z} & + & \mathbf{E}, \\ m \times n & & m \times k & k \times n & & m \times n \end{array}$$

where  $\mathbf{C} = \mathbf{A}(:, J_s)$  is a subset of  $k$  columns of  $\mathbf{A}$ , and  $\mathbf{E}$  is small in magnitude.

**Step 2:** Execute Gram-Schmidt on the *rows* of  $\mathbf{C}$  so that

$$(13) \quad \begin{array}{ccccccc} \mathbf{C} & = & \mathbf{X} & \mathbf{C}(I_s, :), \\ m \times k & & m \times k & k \times k \end{array}$$

where  $I_s$  identifies  $k$  pivot rows in  $\mathbf{C}$ . The factorization (13) is exact since  $\mathbf{C}$  has precisely rank  $k$ . Now combine (12) and (13), observing that  $\mathbf{C}(I_s, :) = \mathbf{A}(I_s, J_s)$  to obtain

$$(14) \quad \begin{array}{ccccccc} \mathbf{A} & = & \mathbf{X} & \mathbf{A}(I_s, J_s) & \mathbf{Z} & + & \mathbf{E}. \\ m \times n & & m \times k & k \times k & k \times n & & m \times n \end{array}$$

**Step 3 — if a CUR is desired:** Set  $\mathbf{R} = \mathbf{A}(I_s, :)$  and  $\mathbf{C} = \mathbf{A}(:, J_s)$  and solve the equation

$$(15) \quad \begin{array}{ccccccc} \mathbf{U} & \mathbf{R} & = & \mathbf{Z} \\ k \times k & k \times n & & k \times n \end{array}$$

for  $\mathbf{U}$  (in a least squares sense). Then combining (12) and (15) we get  $\mathbf{A} \approx \mathbf{CUR}$ .

## Summary of deterministic techniques for computing ID and CUR decompositions

For an  $m \times n$  matrix  $\mathbf{A}$  that is stored in RAM, we can compute structure preserving factorizations as follows:

*Column-ID:* Perform  $k$  steps of Gram-Schmidt on the columns of  $\mathbf{A}$ .

*Row-ID:* Perform  $k$  steps of Gram-Schmidt on the rows of  $\mathbf{A}$ .

*Double-sided-ID:* Perform  $k$  steps of G-S on the columns of  $\mathbf{A}$  to form the ID  $\mathbf{A} \approx \mathbf{CZ}$ . Then perform  $k$  steps of Gram-Schmidt on the rows of the resulting (thin) matrix  $\mathbf{C}$ .

*CUR:* Form a double-sided ID  $\mathbf{A} \approx \mathbf{XA}(I_S, J_S)\mathbf{Z}$ . Then set  $\mathbf{R} := \mathbf{A}(I_S, :)$  and  $\mathbf{C} := \mathbf{A}(:, J_S)$ . Finally solve  $\mathbf{UR} = \mathbf{Z}$  for  $\mathbf{U}$ .

The slow step is in every case the initial  $k$  steps of G-S on the columns/rows of  $\mathbf{A}$ .

**Question:** What if  $\mathbf{A}$  does not fit in RAM? Then Gram-Schmidt becomes impossible, or at least prohibitively slow.

---

***Caveat:** Care must be taken in implementing the algorithms described. Orthonormality must be strictly enforced to avoid problems due to floating point arithmetic.*

## Randomized algorithms for computing ID and CUR decompositions

**Theorem:** Let  $\mathbf{A}$  be an  $m \times n$  matrix of rank  $k$ . Suppose:

(1) We have by some means computed a factorization

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{Y} & \mathbf{W}. \\ m \times n & & m \times k & k \times n \end{array}$$

(2) We have computed a row ID of  $\mathbf{Y}$ , so that

$$\begin{array}{ccccc} \mathbf{Y} & = & \mathbf{X} & \mathbf{Y}(I_S, :). \\ m \times k & & m \times k & k \times k \end{array}$$

Then, *automatically*, we also obtain a row ID for  $\mathbf{A}$ :

$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{X} & \mathbf{A}(I_S, :). \\ m \times k & & m \times k & k \times k \end{array}$$

**Perfect for randomization!** We can find the matrix  $\mathbf{Y}$  using randomized sampling:

1. Draw a Gaussian random matrix  $\mathbf{G}$  of size  $n \times k$ .
2. Form a sample matrix  $\mathbf{Y} = \mathbf{AG}$ .

3. **Do nothing!** You know now that with probability one: 
$$\begin{array}{ccccc} \mathbf{A} & = & \mathbf{Y} & \mathbf{Y}^\dagger \mathbf{A}. \\ m \times n & & m \times k & k \times n \end{array}$$

**Observation 1:** You never need to form the matrix  $\mathbf{Y}^\dagger \mathbf{A}$ .

**Observation 2:** When  $\mathbf{A}$  does not have *exact* rank  $k$ , over-sampling is required.

Recall the randomized ID:

1. Draw a Gaussian random matrix  $\mathbf{G}$  of size  $n \times (k + p)$ .
2. Form a sample matrix  $\mathbf{Y} = \mathbf{AG}$ .
3. Form an ID of the  $n \times (k + p)$  sample matrix:  $[\mathbf{X}, I_s] = \text{ID\_row}(\mathbf{Y}, k)$ .

**Question:** *Why* is it that this strangely simple algorithm works?

Loosely speaking, the reason is that the linear dependencies between the rows of  $\mathbf{Y}$  are the same as the linear dependencies between the rows of  $\mathbf{A}$ .

While not providing a *proof* that the procedure works, it might help intuition to recall that Gaussian random maps preserve distances in expectation. In other words, if  $\mathbf{G}$  is an  $n \times \ell$  Gaussian matrix, and if  $\mathbf{x} \in \mathbb{R}^{1 \times n}$ , then

$$\mathbb{E}[\|\mathbf{xG}\|^2] = \ell \|\mathbf{x}\|^2.$$

As  $\ell$  increases, the probability distribution concentrates tightly around  $\ell \|\mathbf{x}\|^2$ . This implies that, with  $\mathbf{Y} = \mathbf{AG}$ , we have

$$\mathbb{E}[\|\mathbf{Y}(i, :) - \mathbf{Y}(j, :)\|^2] = \ell \|\mathbf{A}(i, :) - \mathbf{A}(j, :)\|^2, \quad \forall i, j \in \{1, 2, \dots, n\}.$$

*Note: This result is a special case of a much stronger theorem presented in R. Vershynin's lecture on Tuesday.*



## ALGORITHM: BASIC RANDOMIZED ID

**Inputs:** An  $m \times n$  matrix  $\mathbf{A}$ , a target rank  $k$ , and an over-sampling parameter  $p$ .

**Outputs:** An  $m \times k$  interpolation matrix  $\mathbf{X}$  and an index vector  $l_s$  such that

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}(l_s, :). \\ m \times n & & m \times k & k \times n \end{array}$$

(1)  $\mathbf{G} = \text{randn}(n, k + p)$ ;

(2)  $\mathbf{Y} = \mathbf{A}\mathbf{G}$ ;

(3) Form an ID of the  $n \times (k + p)$  sample matrix:  $[\mathbf{X}, l_s] = \text{ID\_row}(\mathbf{Y}, k)$ .

This step is executed via Gram-Schmidt on the rows of  $\mathbf{Y}$  (which is thin).

*This is a very simple algorithm that works well when the singular values of  $\mathbf{A}$  decay rapidly. It has complexity  $O(mnk)$  for a dense matrix but is very fast in practice since the only step with  $O(mnk)$  complexity is a single matrix-matrix multiplication.*

## ALGORITHM: BASIC RANDOMIZED ID — ACCURACY ENHANCED

**Inputs:** An  $m \times n$  matrix  $\mathbf{A}$ , a target rank  $k$ , an over-sampling parameter  $p$  (say  $p = 10$ ), and a small integer  $q$  denoting the number of power iterations taken.

**Outputs:** An  $m \times k$  interpolation matrix  $\mathbf{X}$  and an index vector  $l_s$  such that

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}(l_s, :). \\ m \times n & & m \times k & k \times n \end{array}$$

(1)  $\mathbf{G} = \text{randn}(n, k + p)$ ;

(2)  $\mathbf{Y} = \mathbf{A}\mathbf{G}$ ;

(3) **for**  $j = 1 : q$

(4)      $\mathbf{Z} = \mathbf{A}^* \mathbf{Y}$ ;

(5)      $\mathbf{Y} = \mathbf{A}\mathbf{Z}$ ;

(6) **end for**

(7) Form an ID of the  $n \times (k + p)$  sample matrix:  $[\mathbf{X}, l_s] = \text{ID\_row}(\mathbf{Y}, k)$ .

This step is executed via Gram-Schmidt on the rows of  $\mathbf{Y}$  (which is thin).

*The lines in red are new compared to the “basic” scheme.*

*This method provides better accuracy for matrices whose singular values decay slowly.*

*In practice, re-orthonormalization of the sample matrices between iterations may be called for in order to avoid loss of accuracy due to round-off errors.*

## ALGORITHM: FAST RANDOMIZED ID

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

*Outputs:* An  $m \times k$  interpolation matrix  $\mathbf{X}$  and an index vector  $l_s$  such that

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{X} & \mathbf{A}(l_s, :). \\ m \times n & & m \times k & k \times n \end{array}$$

1. Form an  $n \times (k + p)$  SRFT  $\mathbf{R}$ .
2. Form the sample matrix  $\mathbf{Y} = \mathbf{A} \mathbf{R}$ .
3. Form an ID of the  $n \times (k + p)$  sample matrix:  $[\mathbf{X}, l_s] = \text{ID\_row}(\mathbf{Y}, k)$ .

An  $O(mn \log k)$  algorithm for computing an interpolative decomposition of  $\mathbf{A}$ .

## ALGORITHM: FAST RANDOMIZED SVD

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

*Outputs:* Matrices  $\mathbf{U}$ ,  $\mathbf{D}$ , and  $\mathbf{V}$  in an approximate rank- $(k + p)$  SVD of  $\mathbf{A}$ . (I.e.  $\mathbf{U}$  and  $\mathbf{V}$  are orthonormal,  $\mathbf{D}$  is diagonal, and  $\mathbf{A} \approx \mathbf{UDV}^*$ .)

1. Form an  $n \times (k + p)$  SRFT  $\mathbf{R}$ .
2. Form the sample matrix  $\mathbf{Y} = \mathbf{B R}$ .
3. Form an ID of the sample matrix  $\mathbf{Y}$ :  $[\mathbf{X}, I_s] = \text{ID\_row}(\mathbf{Y}, k)$ .
4. Compute the QR decomposition of the interpolation matrix  $[\mathbf{Q}, \mathbf{R}] = \text{qr}(\mathbf{X})$ .
5. Extract  $k + p$  rows of  $\mathbf{A}$ :  $\mathbf{A}^{\text{rows}} = \mathbf{A}(I_s, :)$ .
6. Multiply  $\mathbf{R}$  and  $\mathbf{A}^{\text{rows}}$  to form the  $(k + p) \times n$  matrix  $\mathbf{F} = \mathbf{R A}^{\text{rows}}$ .
7. Decompose the matrix  $\mathbf{F}$  in a singular value decomposition  $[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \text{svd}(\mathbf{F})$ .
8. Form  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ .

An  $O(mn \log k)$  algorithm for computing a partial SVD. If an SVD of exactly rank  $k$  is desired, then truncate the factors computed in Step (7).

**Theory**

## Bound on the expectation of the error for Gaussian test matrices

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  $\mathbf{R}$  denote an  $n \times (k + p)$  Gaussian matrix.

Let  $\mathbf{Q}$  denote the  $m \times (k + p)$  matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{R})$ .

If  $p \geq 2$ , then

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

and

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

*Ref: Halko, Martinsson, Tropp, 2009 & 2011*

## Proofs — Overview:

---

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. Set  $\ell = k + p$ .

Let  $\mathbf{R}$  denote an  $n \times \ell$  “test matrix”, and let  $\mathbf{Q}$  denote the  $m \times \ell$  matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{R})$ .

---

We seek to bound the error  $e_k = e_k(\mathbf{A}, \mathbf{R}) = \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$ , which is a random variable.

1. Make no assumption on  $\mathbf{R}$ . Construct a deterministic bound of the form

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \dots \mathbf{A} \dots \mathbf{R} \dots$$

2. Assume that  $\mathbf{R}$  is drawn from a normal Gaussian distribution.

Take expectations of the deterministic bound to attain a bound of the form

$$\mathbb{E}[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|] \leq \dots \mathbf{A} \dots$$

3. Assume that  $\mathbf{R}$  is drawn from a normal Gaussian distribution.

Take expectations of the deterministic bound conditioned on “bad behavior” in  $\mathbf{R}$  to get that

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \dots \mathbf{A} \dots$$

holds with probability at least  $\dots$ .

## Part 1 (out of 3) — deterministic bound:

---

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. Set  $\ell = k + p$ .

Let  $\mathbf{R}$  denote an  $n \times \ell$  “test matrix”, and let  $\mathbf{Q}$  denote the  $m \times \ell$  matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{R})$ .

---

Partition the SVD of  $\mathbf{A}$  as follows:

$$\mathbf{A} = \mathbf{U} \begin{bmatrix} \mathbf{D}_1 & \\ & \mathbf{D}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* \\ \mathbf{V}_2^* \end{bmatrix} \begin{matrix} k \\ n-k \end{matrix}$$

Define  $\mathbf{R}_1$  and  $\mathbf{R}_2$  via

$$\begin{matrix} \mathbf{R}_1 & = & \mathbf{V}_1^* & \mathbf{R} \\ k \times (k+p) & & k \times n & n \times (k+p) \end{matrix} \quad \text{and} \quad \begin{matrix} \mathbf{R}_2 & = & \mathbf{V}_2^* & \mathbf{R} \\ (n-k) \times (k+p) & & (n-k) \times n & n \times (k+p) \end{matrix}$$

**Theorem:** [HMT2009,HMT2011] Assuming that  $\mathbf{R}_1$  is not singular, it holds that

$$\|\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|\|^2 \leq \underbrace{\|\|\mathbf{D}_2\|\|^2}_{\text{theoretically minimal error}} + \|\|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\|\|^2.$$

Here,  $\|\|\cdot\|\|$  represents either  $\ell^2$ -operator norm, or the Frobenius norm.

*Note:* A similar result appears in Boutsidis, Mahoney, Drineas (2009).



**Recall:**  $\mathbf{A} = \mathbf{U} \begin{bmatrix} \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* \\ \mathbf{V}_2^* \end{bmatrix}$ ,  $\begin{bmatrix} \mathbf{R}_1 \\ \mathbf{R}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{V}_1^* \mathbf{R} \\ \mathbf{V}_2^* \mathbf{R} \end{bmatrix}$ ,  $\mathbf{Y} = \mathbf{A}\mathbf{R}$ ,  $\mathbf{P}$  proj<sup>n</sup> onto  $\text{Ran}(\mathbf{Y})$ .

**Thm:** Suppose  $\mathbf{D}_1 \mathbf{R}_1$  has full rank. Then  $\|\mathbf{A} - \mathbf{P}\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2 \mathbf{R}_2 \mathbf{R}_1^\dagger\|^2$ .

**Proof:** The problem is rotationally invariant  $\Rightarrow$  We can assume  $\mathbf{U} = \mathbf{I}$  and so  $\mathbf{A} = \mathbf{D}\mathbf{V}^*$ .

Simple calculation:  $\|(\mathbf{I} - \mathbf{P})\mathbf{A}\|^2 = \|\mathbf{A}^*(\mathbf{I} - \mathbf{P})^2\mathbf{A}\| = \|\mathbf{D}(\mathbf{I} - \mathbf{P})\mathbf{D}\|$ .

$$\text{Ran}(\mathbf{Y}) = \text{Ran} \left( \begin{bmatrix} \mathbf{D}_1 \mathbf{R}_1 \\ \mathbf{D}_2 \mathbf{R}_2 \end{bmatrix} \right) = \text{Ran} \left( \begin{bmatrix} \mathbf{I} \\ \mathbf{D}_2 \mathbf{R}_2 \mathbf{R}_1^\dagger \mathbf{D}_1^{-1} \end{bmatrix} \mathbf{D}_1 \mathbf{R}_1 \right) = \text{Ran} \left( \begin{bmatrix} \mathbf{I} \\ \mathbf{D}_2 \mathbf{R}_2 \mathbf{R}_1^\dagger \mathbf{D}_1^{-1} \end{bmatrix} \right)$$

Set  $\mathbf{F} = \mathbf{D}_2 \mathbf{R}_2 \mathbf{R}_1^\dagger \mathbf{D}_1^{-1}$ . Then  $\mathbf{P} = \begin{bmatrix} \mathbf{I} \\ \mathbf{F} \end{bmatrix} (\mathbf{I} + \mathbf{F}^* \mathbf{F})^{-1} [\mathbf{I} \ \mathbf{F}^*]$ . (Compare to  $\mathbf{P}_{\text{ideal}} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ .)

Use properties of psd matrices:  $\mathbf{I} - \mathbf{P} \preceq \dots \preceq \begin{bmatrix} \mathbf{F}^* \mathbf{F} & -(\mathbf{I} + \mathbf{F}^* \mathbf{F})^{-1} \mathbf{F}^* \\ -\mathbf{F}(\mathbf{I} + \mathbf{F}^* \mathbf{F})^{-1} & \mathbf{I} \end{bmatrix}$

Conjugate by  $\mathbf{D}$  to get  $\mathbf{D}(\mathbf{I} - \mathbf{P})\mathbf{D} \preceq \begin{bmatrix} \mathbf{D}_1 \mathbf{F}^* \mathbf{F} \mathbf{D}_1 & -\mathbf{D}_1 (\mathbf{I} + \mathbf{F}^* \mathbf{F})^{-1} \mathbf{F}^* \mathbf{D}_2 \\ -\mathbf{D}_2 \mathbf{F} (\mathbf{I} + \mathbf{F}^* \mathbf{F})^{-1} \mathbf{D}_1 & \mathbf{D}_2^2 \end{bmatrix}$

Diagonal dominance:  $\|\mathbf{D}(\mathbf{I} - \mathbf{P})\mathbf{D}\| \leq \|\mathbf{D}_1 \mathbf{F}^* \mathbf{F} \mathbf{D}_1\| + \|\mathbf{D}_2^2\| = \|\mathbf{D}_2 \mathbf{R}_2 \mathbf{R}_1^\dagger\|^2 + \|\mathbf{D}_2\|^2$ .

## Part 2 (out of 3) — bound on expectation of error when $\mathbf{R}$ is Gaussian:

---

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. Set  $\ell = k + p$ .

Let  $\mathbf{R}$  denote an  $n \times \ell$  “test matrix”, and let  $\mathbf{Q}$  denote the  $m \times \ell$  matrix  $\mathbf{Q} = \text{orth}(\mathbf{A}\mathbf{R})$ .

---

**Recall:**  $\|\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|\|^2 \leq \|\|\mathbf{D}_2\|\|^2 + \|\|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\|\|^2$ , where  $\mathbf{R}_1 = \mathbf{V}_1^*\mathbf{R}$  and  $\mathbf{R}_2 = \mathbf{V}_2^*\mathbf{R}$ .

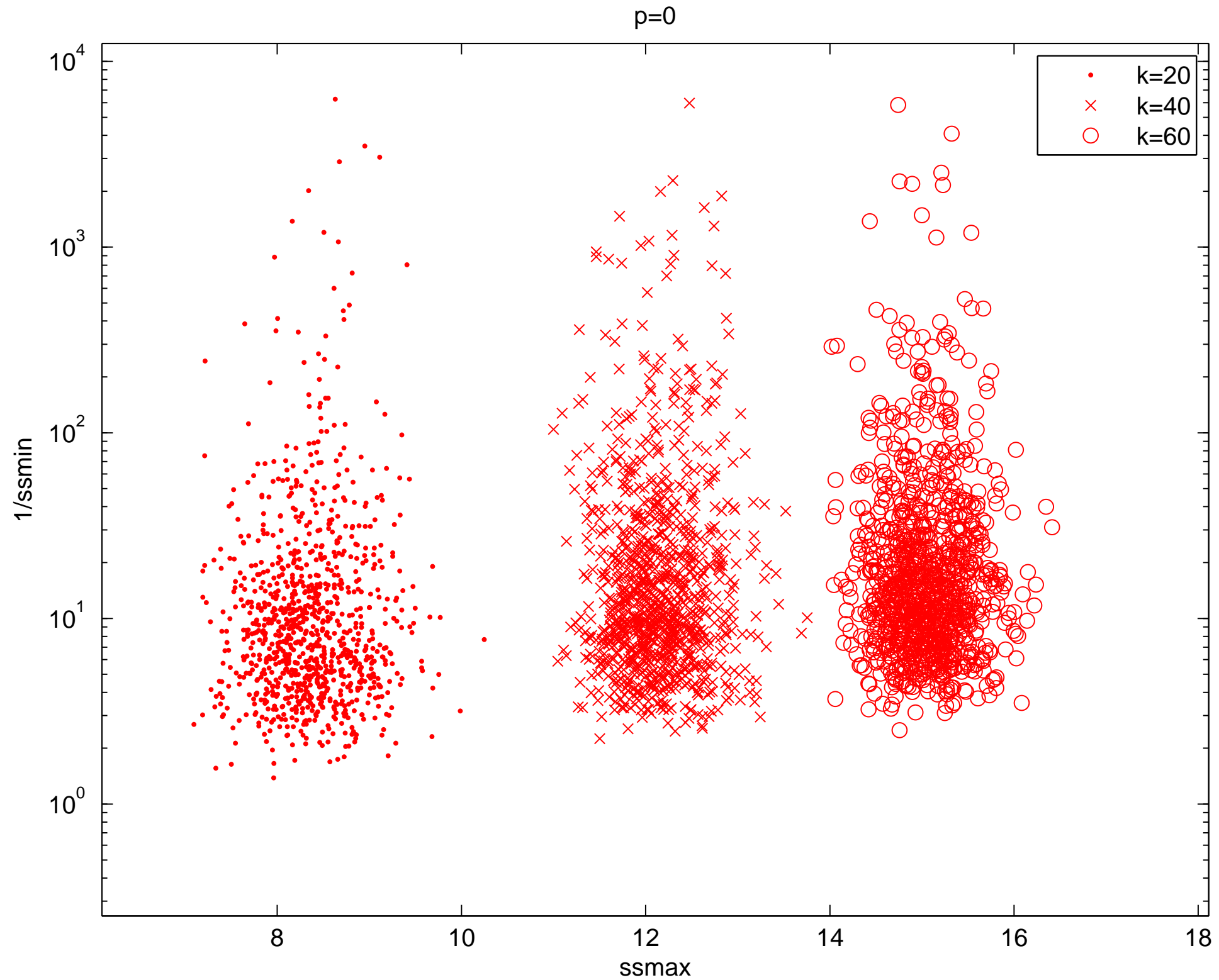
**Assumption:**  $\mathbf{R}$  is drawn from a normal Gaussian distribution.

Since the Gaussian distribution is rotationally invariant, the matrices  $\mathbf{R}_1$  and  $\mathbf{R}_2$  also have a Gaussian distribution. (As a consequence, the matrices  $\mathbf{U}$  and  $\mathbf{V}$  do not enter the analysis and one could simply assume that  $\mathbf{A}$  is diagonal,  $\mathbf{A} = \text{diag}(\sigma_1, \sigma_2, \dots)$ .)

What is the distribution of  $\mathbf{R}_1^\dagger$  when  $\mathbf{R}_1$  is a  $k \times (k + p)$  Gaussian matrix?

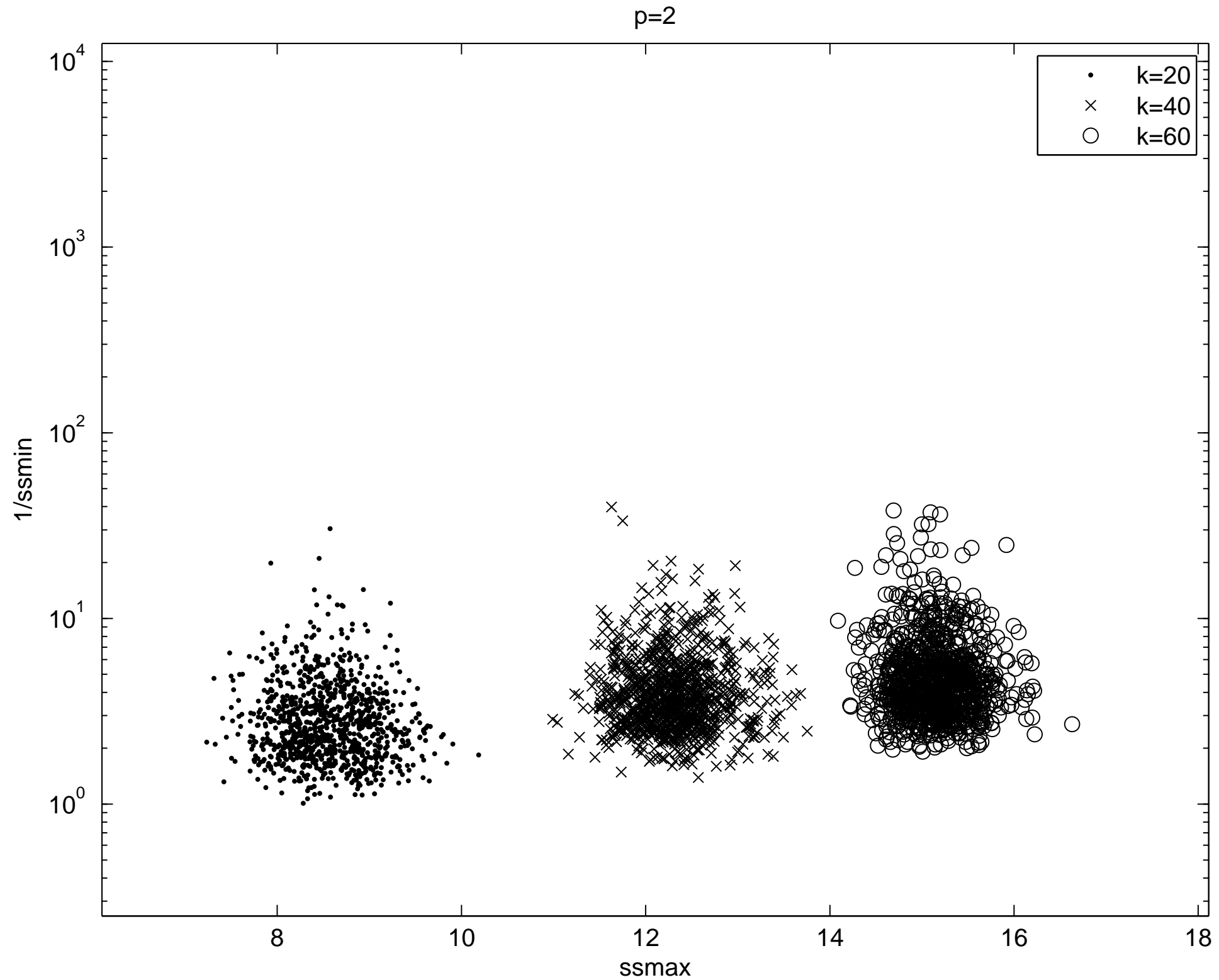
If  $p = 0$ , then  $\|\|\mathbf{R}_1^\dagger\|\|$  is typically large, and is very unstable.

Scatter plot showing distribution of  $1/\sigma_{\min}$  for  $k \times (k + p)$  Gaussian matrices.  $p = 0$



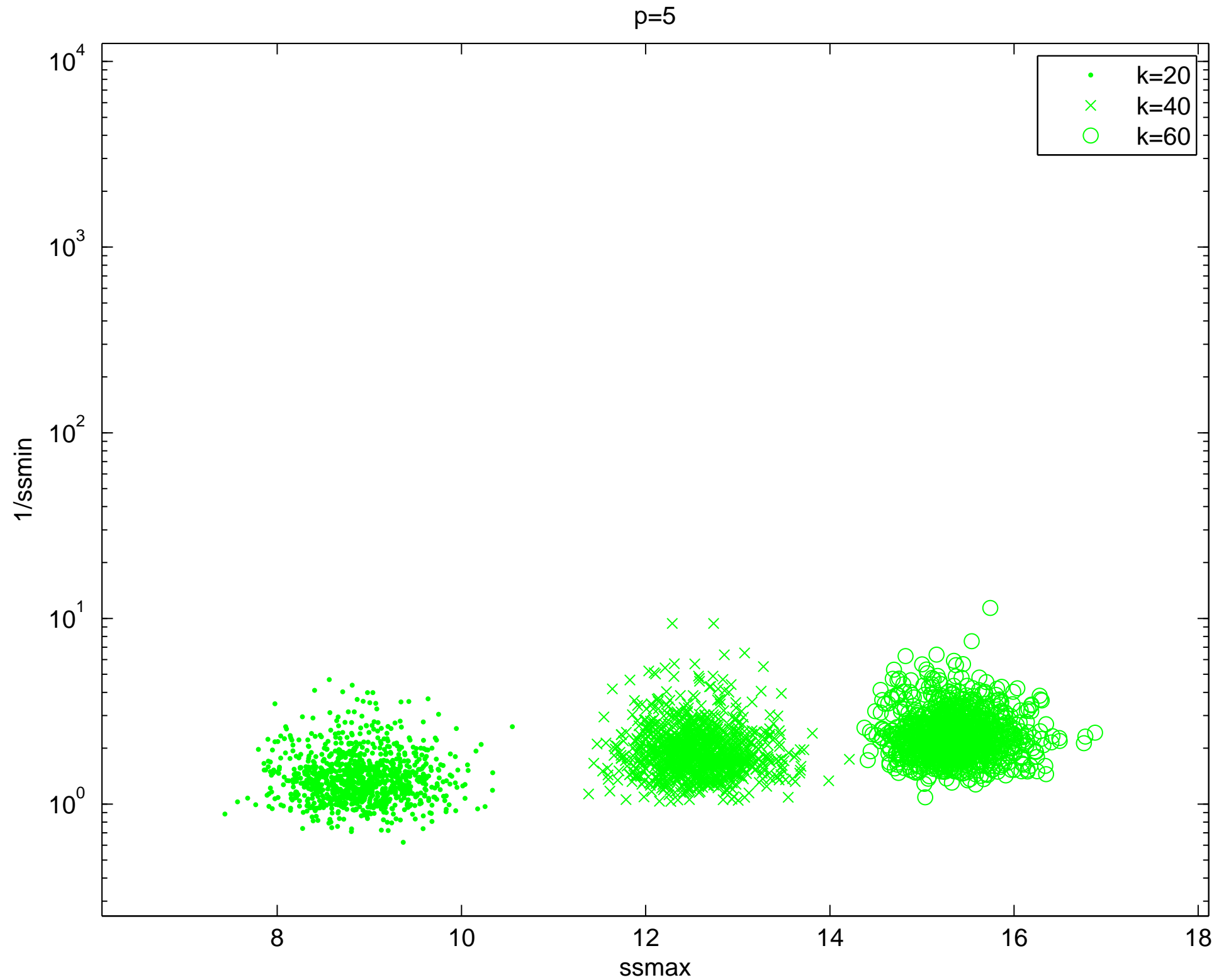
$1/\sigma_{\min}$  is plotted against  $\sigma_{\max}$ .

Scatter plot showing distribution of  $1/\sigma_{\min}$  for  $k \times (k + p)$  Gaussian matrices.  $p = 2$



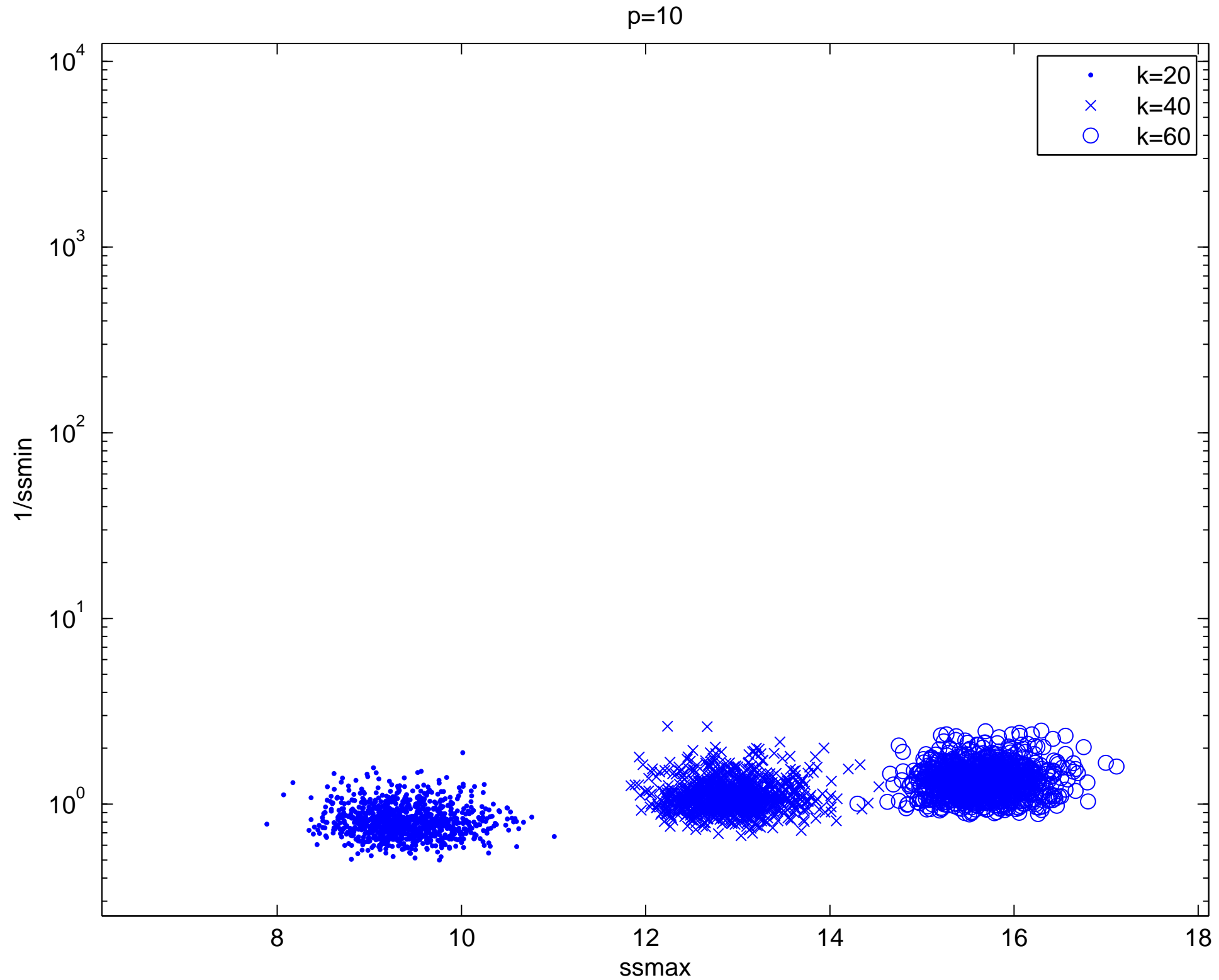
$1/\sigma_{\min}$  is plotted against  $\sigma_{\max}$ .

Scatter plot showing distribution of  $1/\sigma_{\min}$  for  $k \times (k + p)$  Gaussian matrices.  $p = 5$



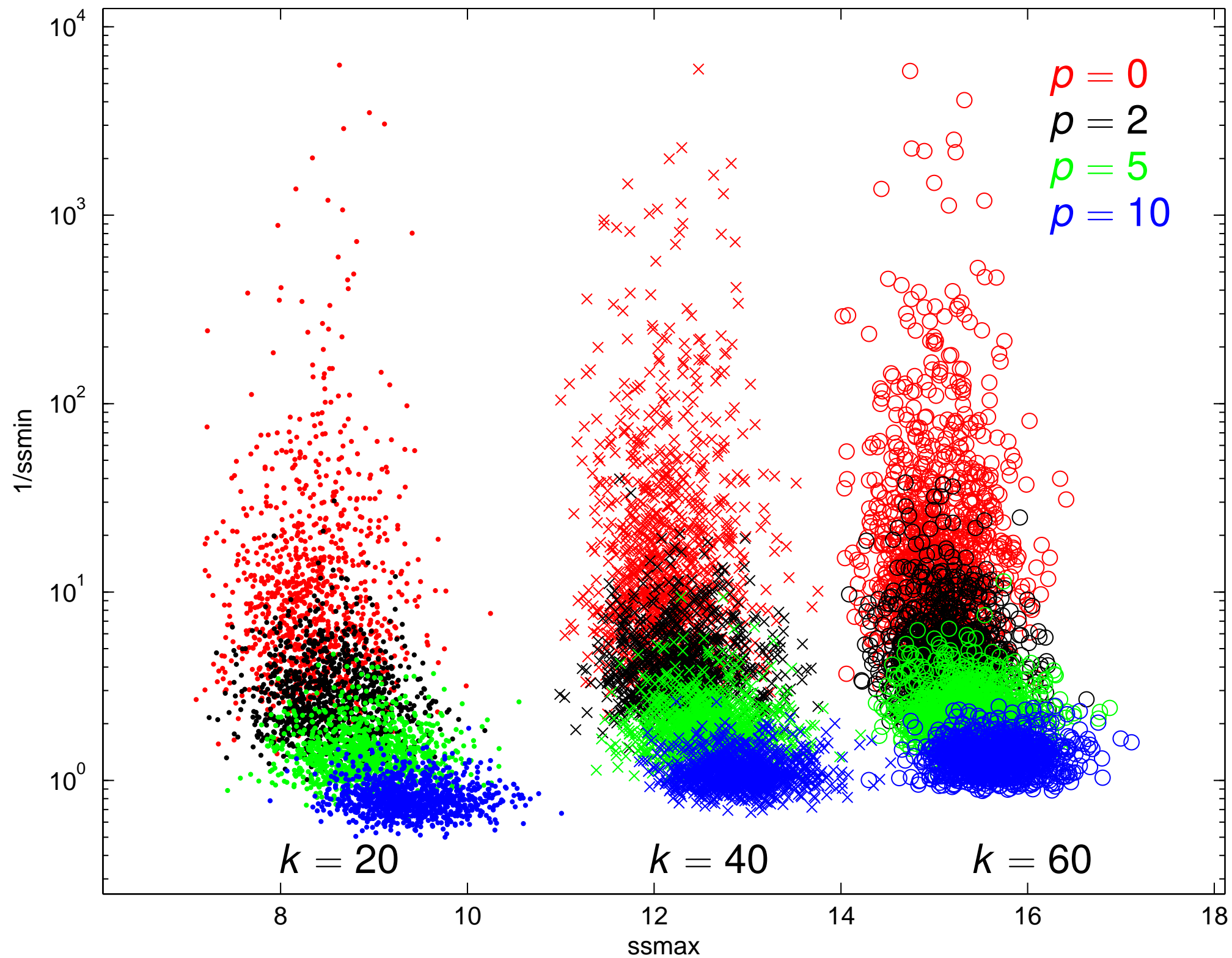
$1/\sigma_{\min}$  is plotted against  $\sigma_{\max}$ .

Scatter plot showing distribution of  $1/\sigma_{\min}$  for  $k \times (k + p)$  Gaussian matrices.  $p = 10$



$1/\sigma_{\min}$  is plotted against  $\sigma_{\max}$ .

Scatter plot showing distribution of  $k \times (k + p)$  Gaussian matrices.



$1/\sigma_{\min}$  is plotted against  $\sigma_{\max}$ .

*Simplistic proof that a rectangular Gaussian matrix is well-conditioned:*

Let  $\mathbf{G}$  denote a  $k \times \ell$  Gaussian matrix where  $k < \ell$ . Let “ $g$ ” denote a generic  $\mathcal{N}(0, 1)$  variable and let “ $r_j$ ” denote a generic random variable distributed like the square root of a  $\chi_j^2$  variable. Then

$$\begin{aligned}
 \mathbf{G} &\sim \begin{bmatrix} g & g & g & g & g & g & \dots \\ g & g & g & g & g & g & \dots \\ g & g & g & g & g & g & \dots \\ g & g & g & g & g & g & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \dots \end{bmatrix} \sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & 0 & \dots \\ g & g & g & g & g & g & \dots \\ g & g & g & g & g & g & \dots \\ g & g & g & g & g & g & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \dots \end{bmatrix} \\
 &\sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & 0 & \dots \\ r_{k-1} & g & g & g & g & g & \dots \\ 0 & g & g & g & g & g & \dots \\ 0 & g & g & g & g & g & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \dots \end{bmatrix} \sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & \dots \\ r_{k-1} & r_{\ell-1} & 0 & 0 & 0 & \dots \\ 0 & g & g & g & g & \dots \\ 0 & g & g & g & g & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots \end{bmatrix} \\
 &\sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & \dots \\ r_{k-1} & r_{\ell-1} & 0 & 0 & 0 & \dots \\ 0 & r_{k-2} & g & g & g & \dots \\ 0 & 0 & g & g & g & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots \end{bmatrix} \sim \dots \sim \begin{bmatrix} r_\ell & 0 & 0 & 0 & 0 & \dots \\ r_{k-1} & r_{\ell-1} & 0 & 0 & 0 & \dots \\ 0 & r_{k-2} & r_{\ell-2} & 0 & 0 & \dots \\ 0 & 0 & r_{k-3} & r_{\ell-3} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots \end{bmatrix}
 \end{aligned}$$

Gershgorin’s circle theorem will now show that  $\mathbf{G}$  is highly likely to be well-conditioned if, e.g.,  $\ell = 2k$ . More sophisticated methods are required to get to  $\ell = k + 2$ .



**Some results on Gaussian matrices.** Adapted from HMT 2009/2011; Gordon (1985,1988) for Proposition 1; Chen & Dongarra (2005) for Propositions 2 and 4; Bogdanov (1998) for Proposition 3.

**Proposition 1:** Let  $\mathbf{G}$  be a Gaussian matrix. Then

$$\begin{aligned} (\mathbb{E} [\|\mathbf{S}\mathbf{G}\mathbf{T}\|_{\mathbb{F}}^2])^{1/2} &\leq \|\mathbf{S}\|_{\mathbb{F}} \|\mathbf{T}\|_{\mathbb{F}} \\ \mathbb{E} [\|\mathbf{S}\mathbf{G}\mathbf{T}\|] &\leq \|\mathbf{S}\| \|\mathbf{T}\|_{\mathbb{F}} + \|\mathbf{S}\|_{\mathbb{F}} \|\mathbf{T}\| \end{aligned}$$

**Proposition 2:** Let  $\mathbf{G}$  be a Gaussian matrix of size  $k \times k + p$  where  $p \geq 2$ . Then

$$\begin{aligned} (\mathbb{E} [\|\mathbf{G}^\dagger\|_{\mathbb{F}}^2])^{1/2} &\leq \sqrt{\frac{k}{p-1}} \\ \mathbb{E} [\|\mathbf{G}^\dagger\|] &\leq \frac{e\sqrt{k+p}}{p}. \end{aligned}$$

**Proposition 3:** Suppose  $h$  is Lipschitz  $|h(\mathbf{X}) - h(\mathbf{Y})| \leq L\|\mathbf{X} - \mathbf{Y}\|_{\mathbb{F}}$  and  $\mathbf{G}$  is Gaussian. Then

$$\mathbb{P}[h(\mathbf{G}) > \mathbb{E}[h(\mathbf{G})] + Lu] \leq e^{-u^2/2}.$$

**Proposition 4:** Suppose  $\mathbf{G}$  is Gaussian of size  $k \times k + p$  with  $p \geq 4$ . Then for  $t \geq 1$ :

$$\begin{aligned} \mathbb{P} \left[ \|\mathbf{G}^\dagger\|_{\mathbb{F}} \geq \sqrt{\frac{3k}{p+1}} t \right] &\leq t^{-p} \\ \mathbb{P} \left[ \|\mathbf{G}^\dagger\| \geq \frac{e\sqrt{k+p}}{p+1} t \right] &\leq t^{-(p+1)} \end{aligned}$$

**Recall:**  $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\|^2$ , where  $\mathbf{R}_1$  and  $\mathbf{R}_2$  are Gaussian and  $\mathbf{R}_1$  is  $k \times k + p$ .

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

**Proof:** First observe that

$$\mathbb{E}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| = \mathbb{E}(\|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\|^2)^{1/2} \leq \|\mathbf{D}_2\| + \mathbb{E}\|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\|.$$

Condition on  $\mathbf{R}_1$  and use Proposition 1:

$$\begin{aligned} \mathbb{E}\|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\| &\leq \mathbb{E}[\|\mathbf{D}_2\| \|\mathbf{R}_1^\dagger\|_F + \|\mathbf{D}_2\|_F \|\mathbf{R}_1^\dagger\|] \\ &\leq \{\text{H\"older}\} \leq \|\mathbf{D}_2\| (\mathbb{E}\|\mathbf{R}_1^\dagger\|_F^2)^{1/2} + \|\mathbf{D}_2\|_F \mathbb{E}\|\mathbf{R}_1^\dagger\|. \end{aligned}$$

Proposition 2 now provides bounds for  $\mathbb{E}\|\mathbf{R}_1^\dagger\|_F^2$  and  $\mathbb{E}\|\mathbf{R}_1^\dagger\|$  and we get

$$\mathbb{E}\|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\| \leq \sqrt{\frac{k}{p-1}} \|\mathbf{D}_2\| + \frac{e\sqrt{k+p}}{p} \|\mathbf{D}_2\|_F = \sqrt{\frac{k}{p-1}} \sigma_{k+1} + \frac{e\sqrt{k+p}}{p} \left(\sum_{j>k} \sigma_j^2\right)^{1/2}.$$

**Some results on Gaussian matrices.** Adapted from HMT2009/2011; Gordon (1985,1988) for Proposition 1; Chen & Dongarra (2005) for Propositions 2 and 4; Bogdanov (1998) for Proposition 3.

**Proposition 1:** Let  $\mathbf{G}$  be a Gaussian matrix. Then

$$\begin{aligned} (\mathbb{E}[\|\mathbf{S}\mathbf{G}\mathbf{T}\|_{\mathbb{F}}^2])^{1/2} &\leq \|\mathbf{S}\|_{\mathbb{F}} \|\mathbf{T}\|_{\mathbb{F}} \\ \mathbb{E}[\|\mathbf{S}\mathbf{G}\mathbf{T}\|] &\leq \|\mathbf{S}\| \|\mathbf{T}\|_{\mathbb{F}} + \|\mathbf{S}\|_{\mathbb{F}} \|\mathbf{T}\| \end{aligned}$$

**Proposition 2:** Let  $\mathbf{G}$  be a Gaussian matrix of size  $k \times k + p$  where  $p \geq 2$ . Then

$$\begin{aligned} (\mathbb{E}[\|\mathbf{G}^\dagger\|_{\mathbb{F}}^2])^{1/2} &\leq \sqrt{\frac{k}{p-1}} \\ \mathbb{E}[\|\mathbf{G}^\dagger\|] &\leq \frac{e\sqrt{k+p}}{p}. \end{aligned}$$

**Proposition 3:** Suppose  $h$  is Lipschitz  $|h(\mathbf{X}) - h(\mathbf{Y})| \leq L\|\mathbf{X} - \mathbf{Y}\|_{\mathbb{F}}$  and  $\mathbf{G}$  is Gaussian. Then

$$\mathbb{P}[h(\mathbf{G}) > \mathbb{E}[h(\mathbf{G})] + Lu] \leq e^{-u^2/2}.$$

**Proposition 4:** Suppose  $\mathbf{G}$  is Gaussian of size  $k \times k + p$  with  $p \geq 4$ . Then for  $t \geq 1$ :

$$\begin{aligned} \mathbb{P}\left[\|\mathbf{G}^\dagger\|_{\mathbb{F}} \geq \sqrt{\frac{3k}{p+1}}t\right] &\leq t^{-p} \\ \mathbb{P}\left[\|\mathbf{G}^\dagger\| \geq \frac{e\sqrt{k+p}}{p+1}t\right] &\leq t^{-(p+1)} \end{aligned}$$

**Recall:**  $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|^2 \leq \|\mathbf{D}_2\|^2 + \|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\|^2$ , where  $\mathbf{R}_1$  and  $\mathbf{R}_2$  are Gaussian and  $\mathbf{R}_1$  is  $k \times k + p$ .

**Theorem:** With probability at least  $1 - 2t^{-p} - e^{-u^2/2}$  it holds that

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \left(1 + t \sqrt{\frac{3k}{p+1}} + ut \frac{e\sqrt{k+p}}{p+1}\right) \sigma_{k+1} + \frac{te\sqrt{k+p}}{p+1} \left(\sum_{j>k} \sigma_j^2\right)^{1/2}.$$

**Proof:** Set  $E_t = \left\{ \|\mathbf{R}_1\| \leq \frac{e\sqrt{k+p}}{p+1}t \text{ and } \|\mathbf{R}_1^\dagger\|_{\text{F}} \leq \sqrt{\frac{3k}{p+1}}t \right\}$ . By Proposition 4:  $\mathbb{P}(E_t^c) \leq 2t^{-p}$ .

Set  $h(\mathbf{X}) = \|\mathbf{D}_2\mathbf{X}\mathbf{R}_1^\dagger\|$ . A direct calculation shows

$$|h(\mathbf{X}) - h(\mathbf{Y})| \leq \|\mathbf{D}_2\| \|\mathbf{R}_1^\dagger\| \|\mathbf{X} - \mathbf{Y}\|_{\text{F}}.$$

Hold  $\mathbf{R}_1$  fixed and take the expectation on  $\mathbf{R}_2$ . Then Proposition 1 applies and so

$$\mathbb{E}[h(\mathbf{R}_2) \mid \mathbf{R}_1] \leq \|\mathbf{D}_2\| \|\mathbf{R}_1^\dagger\|_{\text{F}} + \|\mathbf{D}_2\|_{\text{F}} \|\mathbf{R}_1^\dagger\|.$$

Now use Proposition 3 (concentration of measure)

$$\mathbb{P}\left[\underbrace{\|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\|}_{=h(\mathbf{R}_2)} > \underbrace{\|\mathbf{D}_2\| \|\mathbf{R}_1^\dagger\|_{\text{F}} + \|\mathbf{D}_2\|_{\text{F}} \|\mathbf{R}_1^\dagger\|}_{=\mathbb{E}[h(\mathbf{R}_2)]} + \underbrace{\|\mathbf{D}_2\| \|\mathbf{R}_1^\dagger\|}_{=L} u \mid E_t\right] < e^{-u^2/2}.$$

When  $E_t$  holds true, we have bounds on the “badness” of  $\mathbf{R}_1^\dagger$ :

$$\mathbb{P}\left[\|\mathbf{D}_2\mathbf{R}_2\mathbf{R}_1^\dagger\| > \|\mathbf{D}_2\| \sqrt{\frac{3k}{p+1}}t + \|\mathbf{D}_2\|_{\text{F}} \frac{e\sqrt{k+p}}{p+1}t + \|\mathbf{D}_2\| \frac{e\sqrt{k+p}}{p+1}ut \mid E_t\right] < e^{-u^2/2}.$$

The theorem is obtained by using  $\mathbb{P}(E_t^c) \leq 2t^{-p}$  to remove the conditioning of  $E_t$ .

## ADAPTIVE RANK DETERMINATION

How to proceed when the rank of a matrix is not known in advance.

## Adaptive rank determination — vector-by-vector technique

Let us again start by considering the simplistic case where  $\mathbf{A}$  is *exactly* rank-deficient.

Let  $\mathbf{A}$  be an  $m \times n$  matrix of exact rank  $k$ , where  $k$  is *unknown*.

We seek an  $m \times k$  matrix  $\mathbf{Q}$  whose columns form an ON basis for  $\text{col}(\mathbf{A})$ .

```
 $\mathbf{Q} = [ ];$ 
```

```
for  $i = 1, 2, 3, \dots, ???$ 
```

```
    Draw an  $n \times 1$  Gaussian random vector  $\mathbf{r}_i$ .
```

```
    Compute an  $m \times 1$  sample vector  $\mathbf{y}_i = \mathbf{A}\mathbf{r}_i$ .
```

```
    Project the sample vector away from the basis computed  $\mathbf{z}_i = \mathbf{y}_i - \mathbf{Q}\mathbf{Q}^*\mathbf{y}_i$ .
```

```
    Add the new element to the basis  $\mathbf{Q} = \left[ \mathbf{Q} \frac{\mathbf{z}_i}{\|\mathbf{z}_i\|} \right]$ .
```

```
end for
```

**Observation 1:** While  $i \leq k$ , we know that  $\mathbf{z}_i \neq \mathbf{0}$  with probability 1.

**Observation 2:** Once you come to step  $i = k + 1$ , the vector  $\mathbf{z}_{k+1}$  must be zero!

## Adaptive rank determination — vector-by-vector technique

Let us again start by considering the simplistic case where  $\mathbf{A}$  is *exactly* rank-deficient.

Let  $\mathbf{A}$  be an  $m \times n$  matrix of exact rank  $k$ , where  $k$  is *unknown*.

We seek an  $m \times k$  matrix  $\mathbf{Q}$  whose columns form an ON basis for  $\text{col}(\mathbf{A})$ .

```
Q = [ ];
```

```
for  $i = 1, 2, 3, \dots$ 
```

```
    Draw an  $n \times 1$  Gaussian random vector  $\mathbf{r}_i$ .
```

```
    Compute an  $m \times 1$  sample vector  $\mathbf{y}_i = \mathbf{A}\mathbf{r}_i$ .
```

```
    Project the sample vector away from the basis computed  $\mathbf{z}_i = \mathbf{y}_i - \mathbf{Q}\mathbf{Q}^*\mathbf{y}_i$ .
```

```
    if  $[\mathbf{z}_i = \mathbf{0}]$  then
```

```
        The rank is  $k = i - 1$ .
```

```
        break
```

```
    else
```

```
        Add the new element to the basis  $\mathbf{Q} = [\mathbf{Q} \frac{\mathbf{z}_i}{\|\mathbf{z}_i\|}]$ .
```

```
    end if
```

```
end for
```

## Adaptive rank determination — vector-by-vector technique

Let  $\mathbf{A}$  be an  $m \times n$  matrix whose singular values decay, but do not hit zero.

Let  $\varepsilon > 0$  be a given tolerance. We seek an  $m \times k$  ON matrix  $\mathbf{Q}$  s.t.  $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}} \leq \varepsilon$ .

$\mathbf{Q} = [ ]$ ;

**for**  $i = 1, 2, 3, \dots, ???$

Draw an  $n \times 1$  Gaussian random vector  $\mathbf{r}_i$ .

Compute an  $m \times 1$  sample vector  $\mathbf{y}_i = \mathbf{A}\mathbf{r}_i$ .

Project the sample vector away from the basis computed  $\mathbf{z}_i = \mathbf{y}_i - \mathbf{Q}\mathbf{Q}^*\mathbf{y}_i$ .

Add the new element to the basis  $\mathbf{Q} = \left[ \mathbf{Q} \frac{\mathbf{z}_i}{\|\mathbf{z}_i\|} \right]$ .

**end for**

Observe that

$$\mathbf{z}_i = \mathbf{y}_i - \mathbf{Q}\mathbf{Q}^*\mathbf{y}_i = \mathbf{A}\mathbf{r}_i - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{r}_i = (\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A})\mathbf{r}_i.$$

In consequence, since  $\mathbf{r}_i$  is Gaussian,

$$\mathbb{E}[\|\mathbf{z}_i\|^2] = \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}}^2.$$

**Observation 1:** Once you observe several consecutive  $\mathbf{z}_i$  such that, say,  $\|\mathbf{z}_i\| \leq \varepsilon/2$ , it will “likely” be the case that  $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}} \leq \varepsilon$ .

**Observation 2:** You need to **block** the algorithm for computational efficiency.



## Adaptive rank determination

Let  $\mathbf{A}$  be an  $m \times n$  matrix whose singular values decay, but do not hit zero.

Let  $\varepsilon > 0$  be a given tolerance, and let  $b$  be a “block size.”

We seek an  $m \times k$  ON matrix  $\mathbf{Q}$  s.t.  $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}} \leq \varepsilon$ .

```
Q = [ ];
```

```
for  $i = 1, 2, 3, \dots$ 
```

```
    Draw an  $n \times b$  Gaussian random matrix  $\mathbf{R}_i$ .
```

```
    Compute an  $m \times b$  sample matrix  $\mathbf{Y}_i = \mathbf{A}\mathbf{R}_i$ .
```

```
    Project the sample columns away from the basis computed  $\mathbf{Z}_i = \mathbf{Y}_i - \mathbf{Q}\mathbf{Q}^*\mathbf{Y}_i$ .
```

```
    Orthonormalize the samples  $[\mathbf{Q}_i, \mathbf{R}_i] = \text{qr}(\mathbf{Z}_i, 0)$ . (Unpivoted QR factorization!)
```

```
    if [“several consecutive columns of  $\mathbf{R}_i$  are small”] then
```

```
        Add the appropriate number of columns of  $\mathbf{Q}_i$  to  $\mathbf{Q}$ .
```

```
        break
```

```
    else
```

```
        Add the new element to the basis  $\mathbf{Q} = [\mathbf{Q} \ \mathbf{Q}_i]$ .
```

```
    end if
```

```
end for
```

*Warning: Re-orthogonalization is often needed to combat floating point errors.*

## Adaptive rank determination — with updating

Consider the special case that  $\mathbf{A}$  can be updated, e.g. if it is dense and stored in RAM.

Let  $\mathbf{A}$  be an  $m \times n$  matrix whose singular values decay, but do not hit zero.

Let  $\varepsilon > 0$  be a given tolerance, and let  $b$  be a “block size.”

We seek an  $m \times k$  ON matrix  $\mathbf{Q}$  s.t.  $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}} \leq \varepsilon$ .

```
(1)  $\mathbf{Q} = []$ ;  $\mathbf{B} = []$ ;  
(2) while  $\|\mathbf{A}\| > \varepsilon$   
(3)     Draw an  $n \times b$  Gaussian matrix  $\mathbf{R}_j$ .  
(4)     Compute the  $m \times b$  matrix  $[\mathbf{Q}_j, \sim] = \text{qr}(\mathbf{A}\mathbf{R}_j, 0)$ .  
(5)      $\mathbf{B}_j = \mathbf{Q}_j^*\mathbf{A}$   
(6)      $\mathbf{Q} = [\mathbf{Q} \ \mathbf{Q}_j]$   
(7)      $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_j \end{bmatrix}$   
(8)      $\mathbf{A} = \mathbf{A} - \mathbf{Q}_j\mathbf{B}_j$   
(9) end while
```

A blocked and randomized variation of the classical “modified Gram-Schmidt” algorithm.

*Warning: Re-orthogonalization is often needed to combat floating point errors.*

## Adaptive rank determination — with updating

Consider the special case that  $\mathbf{A}$  can be updated, e.g. if it is dense and stored in RAM.

Let  $\mathbf{A}$  be an  $m \times n$  matrix whose singular values decay, but do not hit zero.

Let  $\varepsilon > 0$  be a given tolerance, and let  $b$  be a “block size.”

We seek an  $m \times k$  ON matrix  $\mathbf{Q}$  s.t.  $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}} \leq \varepsilon$ .

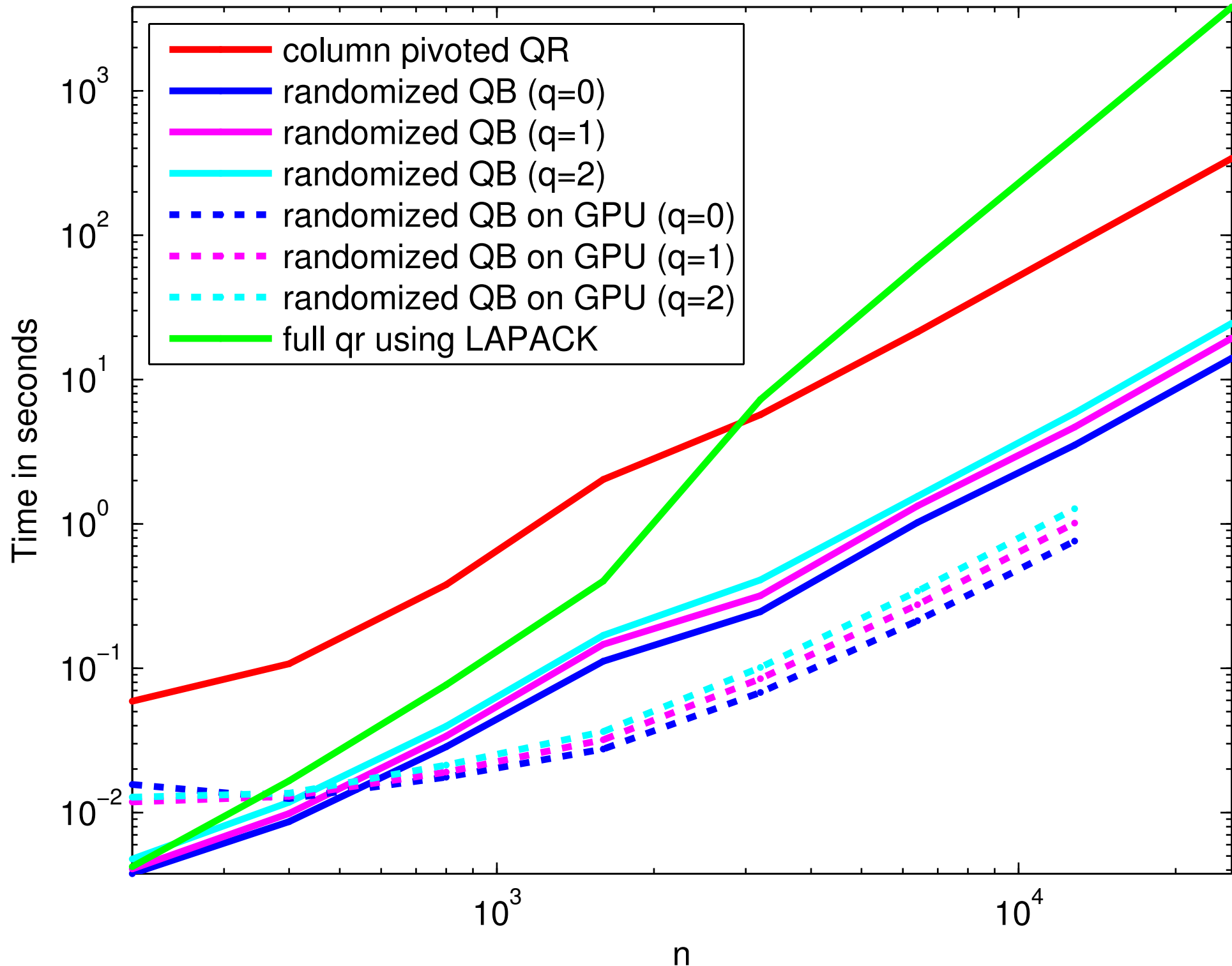
```
(1)  $\mathbf{Q} = [ ]$ ;  $\mathbf{B} = [ ]$ ;  
(2) while  $\|\mathbf{A}\| > \varepsilon$   
(3)     Draw an  $n \times b$  Gaussian matrix  $\mathbf{R}_i$ .  
(4)     Compute the  $m \times b$  matrix  $\mathbf{Q}_i = \text{qr}(\mathbf{A}\mathbf{R}_i, 0)$ .  
(5)      $\mathbf{B}_i = \mathbf{Q}_i^*\mathbf{A}$   
(6)      $\mathbf{Q} = [\mathbf{Q} \ \mathbf{Q}_i]$   
(7)      $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_i \end{bmatrix}$   
(8)      $\mathbf{A} = \mathbf{A} - \mathbf{Q}_i\mathbf{B}_i$   
(9) end while
```

**Observation:** Almost all the work is done by matrix-matrix multiplies.



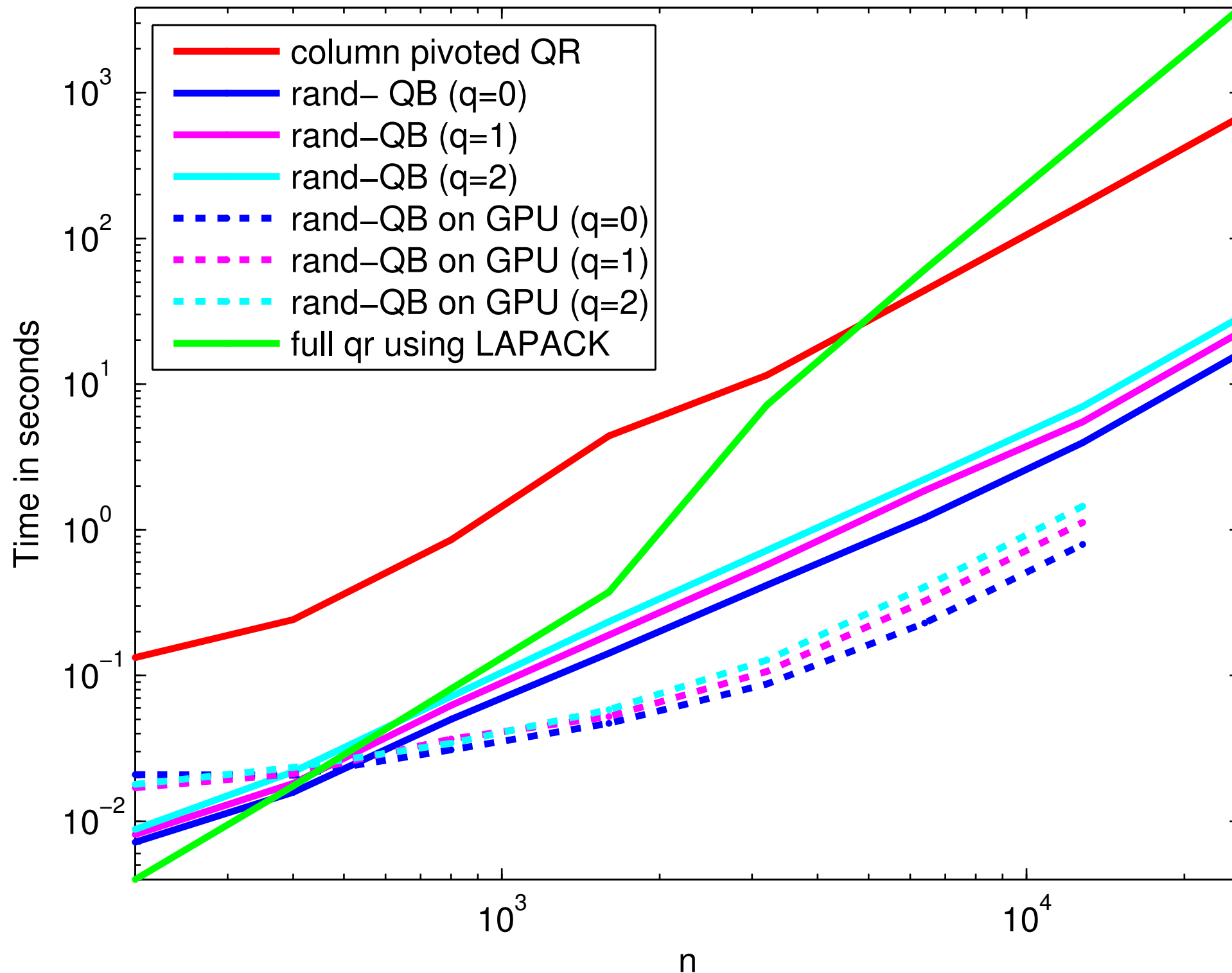
*This algorithm is ideal for running on modern CPUs and GPUs!*

Time for compression of  $n \times n$  matrix.  $k=100$   $kstep=20$



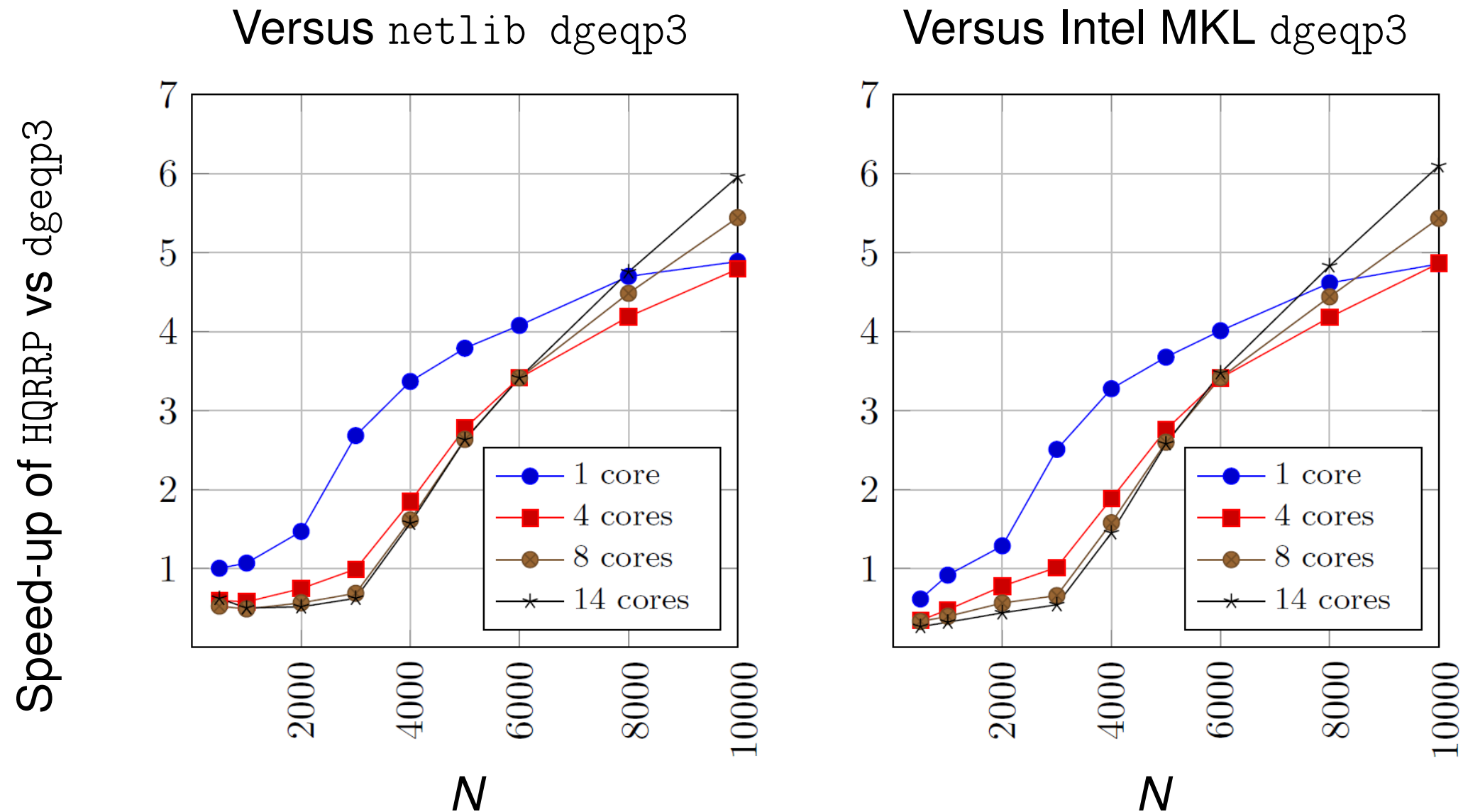
Everything is implemented in Matlab. The “full qr” line refers to Matlab built in qr.  
 CPU = Intel Xeon E-1660 (6 cores, 3.3GHz). GPU = Tesla K40c (2880 cores, 12GB).  
*Caveat: Matlab overhead makes column-pivoted QR slower than it could be.*

Time for compression of  $n \times n$  matrix.  $k=200$   $kstep=40$



Everything is implemented in Matlab. The “full qr” line refers to Matlab built in qr.  
 CPU = Intel Xeon E-1660 (6 cores, 3.3GHz). GPU = Tesla K40c (2880 cores, 12GB).  
*Caveat: Matlab overhead makes column-pivoted QR slower than it could be.*

# Randomized algorithms for FULL factorizations: Column pivoted QR



Speedup attained by our randomized algorithm *HQRQP* 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 pre-conditioners

**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 pre-conditioner:*

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.*

It is often advantageous to add a final step of *à posteriori error estimation.*

This can typically be done very cheaply using randomized sampling.

# Example 1 of two-stage approach: Randomized SVD

---

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

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

where  $\mathbf{U}$  and  $\mathbf{V}$  are orthonormal, and  $\mathbf{D}$  is diagonal.

---

(A) *Randomized pre-conditioner:*

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.

Observe that distortions in the randomized projections are fine, since all we need is a subspace that captures “most” of the range. Pollution from unwanted singular modes is harmless, as long as we capture the dominant ones. The risk of missing the dominant ones is for practical purposes zero.



# Example 1 of two-stage approach: Randomized SVD

---

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

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

where  $\mathbf{U}$  and  $\mathbf{V}$  are orthonormal, and  $\mathbf{D}$  is diagonal.

---

Fix an over-sampling parameter  $p$ . Say  $p = 10$ .

## (A) *Randomized pre-conditioner:*

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

$$\mathbf{G} = \text{randn}(n, k+p)$$

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

$$\mathbf{Y} = \mathbf{A} * \mathbf{G}$$

A.3 Form an  $m \times (k + p)$  orthonormal matrix  $\mathbf{Q}$  such that  $\mathbf{Y} = \mathbf{Q} \mathbf{R}$ .

$$[\mathbf{Q}, \mathbf{R}] = \text{qr}(\mathbf{Y})$$

## (B) *Deterministic post-processing:*

B.1 Form the  $(k + p) \times n$  matrix  $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ .

$$\mathbf{B} = \mathbf{Q}' * \mathbf{A}$$

B.2 Form SVD of the matrix  $\mathbf{B}$ :  $\mathbf{B} = \hat{\mathbf{U}} \mathbf{D} \mathbf{V}^*$ .

$$[\mathbf{Uhat}, \mathbf{Sigma}, \mathbf{V}] = \text{svd}(\mathbf{B}, 0)$$

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

$$\mathbf{U} = \mathbf{Q} * \mathbf{Uhat}$$

(Truncate the last  $p$  terms in step B.2 to attain a factorization of precise rank  $k$ .)

## Example 2 of two-stage approach: Nearest neighbor search in $\mathbb{R}^D$

Peter Jones, Andrei Osipov, Vladimir Rokhlin

---

**Objective:** Suppose you are given  $n$  points  $\{\mathbf{x}_j\}_{j=1}^n$  in  $\mathbb{R}^D$ . The coordinate matrix is

$$\mathbf{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \cdots \ \mathbf{x}_n] \in \mathbb{R}^{D \times n}.$$

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

---

If  $D$  is “small” (say  $D \leq 10$  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  $D$  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.

## Example 2 of two-stage approach: Nearest neighbor search in $\mathbb{R}^D$

Peter Jones, Andrei Osipov, Vladimir Rokhlin

---

**Objective:** Suppose you are given  $n$  points  $\{\mathbf{x}_j\}_{j=1}^n$  in  $\mathbb{R}^D$ . The coordinate matrix is

$$\mathbf{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \cdots \ \mathbf{x}_n] \in \mathbb{R}^{D \times n}.$$

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

---

### (A) *Randomized probing of data:*

Use a Johnson-Lindenstrauss random projection to map the  $n$ -particle problem in  $\mathbb{R}^D$  (where  $D$  is large) to an  $n$ -particle problem in  $\mathbb{R}^d$  where  $d \sim \log n$ . Run a deterministic nearest-neighbor search in  $\mathbb{R}^d$  and store a list of the  $\ell$  nearest neighbors for each particle (for simplicity, one can set  $\ell = k$ ). Then repeat the process several times. If for a given particle a previously undetected neighbor is discovered, then simply add it to a list of potential neighbors.

### (B) *Deterministic post-processing:*

The randomized probing will result in a list of putative neighbors that typically contains more than  $k$  elements. But it is now easy to compute the pairwise distances in the original space  $\mathbb{R}^D$  to judge which candidates in the list are the  $k$  nearest neighbors.

## An observation:

The randomized methods presented are close in spirit to randomized algorithms such as:

- Randomized quick-sort.  
(With variations: computing the median / order statistics / *etc.*)
- Routing of data in distributed computing with unknown network topology.
- Rabin-Karp string matching / verifying equality of strings.
- Verifying polynomial identities.

Many of these algorithms are of the type that it is the *running time* that is stochastic.

The quality of the final output is excellent.

The randomized algorithm that is perhaps the best known within numerical analysis is [Monte Carlo](#). This is somewhat lamentable given that MC is often a “last resort” type algorithm used when the curse of dimensionality hits — inaccurate results are tolerated simply because there are no alternatives.


(These comments apply to the traditional “unreformed” version of MC — for many applications, more accurate versions have been developed.)

## Concluding remarks:

- For large scale SVD/PCA of dense matrices, these algorithms are highly recommended; they compare favorably to existing methods in almost every regard.
- The approximation error is a random variable, but its distribution is tightly concentrated. Rigorous error bounds that are satisfied with probability  $1 - \eta$  where  $\eta$  is a user set “failure probability” (e.g.  $\eta = 10^{-10}$ ).
- Randomized methods are even very competitive for *full* factorizations! (New!)
- We presented two instantiations of a “two-stage” algorithmic template:  
*Stage A:* Randomized methods to develop a sketch of the data. “Where to look.”  
*Stage B:* Highly accurate classical methods, operating on original data.  
My guess: This two-stage approach is likely to find more uses!
- These lectures mentioned *error estimators* only briefly, but they are important. Can operate independently of the algorithm for improved robustness. Typically cheap and easy to implement. Used to determine the actual rank.
- The theory can be hard (at least for me), but *experimentation is easy!* Concentration of measure makes the algorithms behave as if deterministic.

## Lecture notes:

- Summary of lectures at: [http://amath.colorado.edu/faculty/martinss/2016\\_PCMI/](http://amath.colorado.edu/faculty/martinss/2016_PCMI/)  
Provides pointers to original papers and additional reading.
- Book manuscript:  
*Randomized methods for matrix computations and analysis of high dimensional data*  
A draft of first half available at: <http://arxiv.org/abs/1607.01649>

 Feedback, suggestions, errata, etc, are very welcome!

## 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)

## Papers:

- *Original paper:* P.G. Martinsson, V. Rokhlin, and M. Tygert, “A randomized algorithm for the approximation of matrices”. 2006 report YALEU/DCS/RR-1361, 2011 paper in ACHA.
- *Survey:* N. Halko, P.G. Martinsson, J. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.” *SIAM Review*, 2011.
- Much much more! Many authors. Several excellent surveys. See book manuscript and resources at: [http://amath.colorado.edu/faculty/martinss/2016\\_PCMI/](http://amath.colorado.edu/faculty/martinss/2016_PCMI/)